diveintocoffeescript Documentation

Release 0.1.0

Eugene Mirotin

June 25, 2011

Contents

0. Introduction and Assumptions

The book follows the great concept of 'diving into' introduced by Mark Pilgrim in his excellent Dive Into Python. This means that normally chapters start with the complete working code example, which is then explained and discussed.

The book rewrites the majority of Mark's sample programs in CoffeeScript language using the growing Node.js infrastructure.

Though it slowly starts with the very basic ideas, you have to have some background in order to understand what's going on.

A note on operating system

The majority of things covered in the book *should* work on Windows. Though nobody guarantees it.
I'm working on Mac OSX, a member of *nix family, and thus almost everything will work on Linux in exactly the same way as here.
Shell sessions are shown in bash.
For Windows users, where I expect differences or problems, I provide links to external resources.

1.1 Resources and Pointers

Here are some facts and titles. If you feel uncomfortable with some of them, follow the links and read through the articles and definitions.

Before you move to the next chapter, become sure that you think I'm an idiot telling you obvious things.

1.1.1 Internet, etc.

Hope, you are fine with this. Internet is a distributed system consisting of thousands of web-servers.

Everything in the Internet basically follows the client-server oriented architecture, which means that some nodes (systems, computers) - servers - produce *content*, and others - clients - request and consume it.

WWW (World Wide Web) works over Internet to deliver hypertext documents.

In case of WWW servers are Web servers, clients are (normally) the browser programs, and content is transfered (mostly) over the HTTP protocol.

Hypertext initially meant to be a text with inter-links (like between encyclopedia articles).

1.1.2 HTML

Hypertext format used by WWW is HTML.

HTML (HyperText Markup Language) is a set of rules for specially marking text files to be interpreted by browsers as hypertext documents.

HTML mixes tags into text content, outlining its structure and allowing styling.

If you don't know HTML, follow this tutorial: http://www.w3schools.com/html/default.asp, specifically these chapters:

- Introduction, Get Started
- Basics, Elements, Attributes,
- Headings, Paragraphs, Formatting, Links, *Images <http://www.w3schools.com/html/html_images.asp>*, Lists
- HTML Layout, Doctypes (*this section has some historycal meaning*), Head,
- Summary.

HTML is not mandatory for early parts of the book, but I think understanding Web, HTTP and HTML is mandatory before you start learning advanced topics like Node.js and CoffeeScript (even though these advanced topics start with examples that are much simpler than HTML itself).

The most recent / modern HTML standard version is HTML5. This tutorial gives an overview of the changes done since HTML4: http://www.w3schools.com/html5/default.asp.

There is also a book by Mark Pilgrim (sic!) titled 'Dive Into HTML5' (sic!), full of detailed explanations and historical flashbacks giving reasoning and background for this evolving standard.

It's not mandatory to know HTML5 (its mostly straightforward for the reader familiar with HTML4), but if you want to be in Web development, you will have to learn it anyway very soon.

1.1.3 CSS

As HTML grew out of simple hypertext documents to full-featured **web pages**, a need for non-obtrusive separated styling arouse.

This was solved with CSS. CSS, standing for Cascade Style Sheets (very confusing title, actually) allows setting specific visual and layout properties for HTML elements.

Follow this tutorial to understand its basics: http://www.w3schools.com/css/default.asp.

You don't need deep understanding of CSS to read through this book, but basics are recommended.

Understanding CSS selectors is also the must for working with jQuery library.

1.1.4 JavaScript

Once developers understanded they want some *dynamics* in their pages, a JavaScript language appeared (read here for short historical insight: http://en.wikipedia.org/wiki/JavaScript#History).

What you should know about JavaScript:

- it has **nothing** to do with Java. At all. Read here for some highlights,
- it was specially designed to work in the browser,
- the biggest strength of JavaScript (at the moment of appearance) was that it could interact with DOM - Document Object Model. DOM is the way the HTML document structure (as stored and manipulated by the browser) is exposed to the JavaScript code,
- JavaScript basics: http://javascript.crockford.com/survey.html,

- JavaScript tutorial: http://www.w3schools.com/js/default.asp,
- (a quite short) JavaScript code style guide: part one, part two,
- a more detailed style guide: http://javascript.crockford.com/code.html,
- JavaScript is actually the world's most popular programming language: http://javascript.crockford.com/popular.html.

Make sure you understand JavaScript, its purposes and main coding blocks before we move on.

1.1.5 Node.js

As you should already know, JavaScript was the language created for Web *client-side* programming - manipulating retrieved content as opposed to generating the content on the server.

That means every dynamic non-trivial web site had to be written in at least 4 languages: HTML for document structure and markup, CSS for document styling and layout, server-side language like PHP to put dynamic data (like those retieved from the database) into the page, and JavaScript to add client-side dynamics, for example, perform in-place data validations.

Since that time several attempts to move JavaScript out of the browser were done, including Rhino, a Java implementation allowing JS execution from the command line. It didn't get big success, I suppose mostly because of performance issues.

In 2009 a project called CommonJS was started aimed at specifying some standards for wider JS usage, from GUI apps to web servers.

Then, Google released its browser, Google Chrome, shipped with the new JS engine - V8.

Based on V8, a Node.js was created - a wrapping project, using V8 as JS engine, and implementing CommonJS library.

So, Node.js:

- is a JavaScript imeplementation not tied to the browser,
- uses the V8 JS engine, considered to be really fast and well performing,
- implements a set of useful common libraries, such as filesystem access, HTTP client (agent) ans server, cryptographic primitives (through OpenSSL bindings), etc.
- allows command-line JS execution. It is often said that Node.js is a server-side Javascript. Well, yes, but it doesn't mean it can only be used for writing HTTP servers - it rather means it is executed as a normal program, not inside a browser,
- implements Console API used by many browser JS debuggers (like Firebug or Chrome developer tools),
- is asyncronous event-driven system, which means it heavily uses callback idiom for almost every operation that may require significant time for completion,
- got lots of modules written, providing impressive set of functionality for writing web servers and service of any kind.

1.1.6 CoffeeScript

CoffeeScript (*you don't have to read through its documentation right now*) is a rethinking of JS *syntax*. Technically its a *grammar* (a set of rules for writing coffee files), and a *program* translating the coffee files into js files.

So, everything you can do with JS can be done with CoffeeScript, and vice versa - every CoffeeScript construct is just a synonim for a JS function, method or common idiom.

Why another language? Well, CoffeeScript is much more beautiful and laconic. It also incorporates some best practices, like global closures, === and !== usage, etc.

The code written in CoffeeScript is normally shorter, more readable and maintainable, and usually better than the same code manually written in JS.

CoffeeScript is based on Node.js, which means that you need Node.js to transform CoffeeScript code to JavaScript. Alternatively there exist a way to directly use CoffeeScript in your web pages, though it is not recommended for production usage.

1.2 Summary

HTML documents are requested by web browsers from the web servers over the Internet HTTP protocol.

CSS and JS can be shipped together with them, and are interpreted by web browsers to add styling and dynamical behavior to the web pages.

JS jumped out of the browser box and can now be used as a stand-alone or server-side language thanks to Node.js.

Node.js uses fast V8 JS engine, implements common library, is based on asynchronous operations and events paradigms, and has lots of useful extra modules.

CoffeeScript changes JS syntax, and is based on Node.js.

1. Installing Node.js and CoffeeScript

First of all, make sure you have your favorite text editor, and you terminal ready.

2.1 Node.js

Node.js can be installed on any platform - easier on *nix, harder on Windows.

See short official instructions, and more detailed from howtonode.org.

On Mac OSX with MacPorts you can also do

```
sudo port install nodejs
```

Now let's test it:

```
$ node --version
v0.4.8
```

In your case version number may differ, as Node is constantly developed and updated.

Let's create a test file hello.js:

```
1 console.log('Hellow, World!')
```

and run it:

```
$ node hello.js
Hello, World!
```

This was actually our first Node.js program. Let's explain it:

1) console.log is a function that outputs given text to the terminal from which the Node program was started,

1') in JS lines of text (strings) are marked with the single (') or double (") quotes.

2.2 NPM

NPM stands for Node Package Manager, and is the interface to the centralized storage of Node.js modules.

If you have *curl*, you can

```
$ curl http://npmjs.org/install.sh | sh
```

Read here if you encounter any problems.

2.3 CoffeeScript

Having *npm* you can install CoffeeScript as simple as

```
npm install -g coffee-script
```

You may need root priveledges, or running it as

```
sudo npm install -g coffee-script
```

Now, let's test it:

```
$ coffee --version
CoffeeScript version 1.1.1
```

Finally, let rewrite our *hello.js* in CoffeeScript:

```
1 console.log 'Hellow, World!'
```

and run it:

```
$ coffee hello.coffee
Hello, World!
```

This was our first CoffeeScript program. Let's see what changed, and what didn't:

0) Changed: we are now executing it with *coffee* instead of *node*,

1) Same: we use the same console.log function, because CoffeeScript only changes language *syntax*,

1') Same: in CoffeeScript strings are also marked with the single (') or double (") quotes,

1'') Changed: in CoffeeScript parenthesis are optional for function calls. Sometimes it makes the calls much more beautiful and readable (not in this case, though).

CHAPTER 3

Indices and tables

- *genindex*
- *search*