Содержание

Предисловие

Моя профессиональная карьера разработчика началась в 1999 году, когда я получил первую зарплату как разработчик. (Я не считаю несколько лет, когда я просто получал удовольствие, играя с Веб.) В 1999 году Всемирная паутина была жутким местом. HTML-файлы были перегружены тегами `font` и `table`. Каскадные таблицы стилей CSS только-только начали выходить на сцену. Язык JavaScript [1] существовал всего несколько лет, а война браузеров была в самом разгаре. Безусловно, тогда можно было написать JavaScript-сценарий, выполняющий некоторые операции в одном браузере, но смог бы он работать в другом? Скорее всего, нет. Из-за этого в 2000 годах язык JavaScript получил дурную славу.

В середине 2000-х произошло два важных события, которые помогли JavaScript подняться в глазах разработчиков. Первым из них было появление технологии AJAX. [2] Технология AJAX позволяет разработчикам создавать более быстрые и более интерактивные вебстраницы, благодаря возможности отправлять запросы на сервер в фоновом режиме и устранению необходимости для конечного пользователя обновлять содержимое окна браузера.

Вторым событием стало появление популярных библиотек на JavaScript, таких как Prototype, [3] которые существенно упростили создание JavaScript-сценариев, совместимых со всеми типами браузеров. Появилась возможность использовать технологию AJAX, чтобы сделать приложения более интерактивными и удобными в использовании, и задействовать библиотеку, такую как Prototype, чтобы обеспечить совместимость с основными типами браузеров.

В 2010 году, а точнее в 2011, развитие Всемирной паутины пошло по пути создания «одностраничных» приложений. Такие приложения выполняются под управлением JavaScript-фреймворков, таких как Backbone.js. [4] Эти фреймворки позволяют применять шаблон проектирования MVC [5] с использованием JavaScript. Стало возможным писать на JavaScript целые приложения, а затем загружать их и выполнять в браузере конечного пользователя. Все вместе это

позволяет писать поразительно интерактивные и полнофункциональные клиентские приложения.

Однако, с точки зрения разработчика, ситуация выглядела не так радужно. Несмотря на то, что фреймворки и инструменты значительно упростили разработку подобных приложений, сам язык JavaScript оставался болезненным местом. Язык JavaScript является одновременно невероятно мощным, и в то же время чрезвычайно запутанным. Он полон парадоксов и ловушек, которые быстро могут сделать ваш программный код неконтролируемым и наполненным ошибками.

Так чего же хотят разработчики? Они хотят создавать эти замечательные новые приложения, но единственным языком, который понимают все браузеры, является JavaScript. Конечно, они могут писать эти приложения на Flash, [6] но для этого в браузеры необходимо устанавливать расширения, к тому же эти расширения отсутствуют для некоторых платформ, таких как устройства на iOS [7].

Впервые с языком CoffeeScript [8] я столкнулся в октябре 2010 года. CoffeeScript давал мне надежду приручить JavaScript и подчеркивал наиболее выгодные стороны замысловатого языка, каковым является JavaScript. Он имеет ясный синтаксис, отдавая предпочтение пробелам вместо знаков пунктуации, и защищает от ловушек, поджидающих JavaScript-разработчиков на каждом шагу, таких как неочевидные правила видимости и неправильное употребление операторов сравнения. Но самое замечательное, что в конечном итоге программный код на CoffeeScript компилируется в стандартный программный код на JavaScript, который может выполняться в любом браузере или в другой среде выполнения JavaScript.

Когда я впервые попробовал использовать CoffeeScript, язык был еще далек от совершенства, даже в версии 0.9.4. Я использовал его в проекте моего клиента, чтобы попробовать и увидеть, является ли правдой все, что я слышал о нем. К сожалению, две причины заставили меня отложить его в сторону. Во-первых, язык еще не был готов к широкому использованию. В нем было слишком много ошибок и в нем отсутствовали многие возможности.

Вторая причина, заставившая меня отказаться от CoffeeScript, заключалась в том, что приложение, на котором я проводил эксперименты, не было настоящим JavaScript-приложением. Мне требовалось реализовать лишь кое-какие проверки и организовать отправку запросов с использованием технологии AJAX, что, благодаря

помощи Ruby on Rails [9] достигалось совсем небольшим объемом программного кода на JavaScript.

Так что же заставило меня вернуться к CoffeeScript? Спустя примерно шесть месяцев после первого знакомства с CoffeeScript, было объявлено, [10] что Rails 3.1 будет распространяться вместе с CoffeeScript, в качестве механизма JavaScript по умолчанию. Как и большинство разработчиков, я был ошеломлен этой новостью. Я пытался использовать язык CoffeeScript и не считал его чем-то выдающимся. О чем они думали?

В отличие от большинства моих собратьев разработчиков я решил уделить время, чтобы по-новому взглянуть на CoffeeScript. Шесть месяцев – достаточно долгий срок в разработке любого проекта. Язык CoffeeScript проделал длинный, очень длинный путь. И я решил еще раз попробовать использовать его, на этот раз в приложении, содержащем достаточно большой объем программного кода на JavaScript. Спустя несколько дней повторного использования CoffeeScript, я не только изменил свои взгляды, но и полюбил этот язык.

Не могу сказать точно, что повлияло на мои убеждения, и не буду пытаться объяснить, почему я полюбил этот язык. Я хочу, чтобы вы сами сформировали свое мнение о нем. Надеюсь, что в ходе чтения этой книги вы не только станете приверженцами, но и активными сторонниками этого замечательного маленького языка по вашим собственным причинам. А я коротко расскажу вам о том, что ждет вас впереди. Ниже приводится небольшой фрагмент программного кода на CoffeeScript из действующего приложения, а вслед за ним – эквивалентный фрагмент на JavaScript. Наслаждайтесь!

Пример: (исходный файл: `sneak_peak.coffee`)

```
@updateAvatars = ->
    names = $('.avatar[data-name]').map -> $(this).data('name')
      Utils.findAvatar(name) for name in $.unique(names)
```

Пример: (исходный файл: `sneak_peak.js`)

```
(function() {
    this.updateAvatars = function() {
        var name, names, _i, _len, _ref, _results;
        names = $('.avatar[data-name]').map(function() {
            return $(this).data('name');
        });
        _ref = $.unique(names);
```

```
        _results = [];
        for (_i = 0, _len = _ref.length; _i < _len; _i++) {
            name = _ref[_i];
            _results.push(Utils.findAvatar(name));
        }
        return _results;
    };
}).call(this);
```

Что такое CoffeeScript?

CoffeeScript – это язык программирования, программный код на котором компилируется в программный код на языке JavaScript. Не слишком понятно, знаю, но это так и есть. Язык CoffeeScript близко напоминает такие языки программирования, как Ruby [11] и Python. [12] Он создавался с целью помочь разработчикам повысить производительность труда при разработке программного кода на JavaScript. За счет избавления от необходимости использовать знаки пунктуации, такие как скобки, точки с запятой, и прочие, и использования значимых пробелов взамен этих символов, вы сможете быстро сосредотачиваться на программном коде, а не на бесконечных проверках – расставлены ли все закрывающие фигурные скобки.

Представьте, что вы пишите такой код на JavaScript:

Пример: (исходный файл: `punctuation.js`)

```
(function() {
    if (something === something_else) {
        console.log('do something');
    } else {
        console.log('do something else');
    }
}).call(this);
```

Почему бы не записать его так:

Пример: (исходный файл: `punctuation.coffee`)

```
if something is something_else
    console.log 'do something'
else
    console.log 'do something else'
```

Кроме того, язык CoffeeScript предоставляет ряд сокращенных форм записи, позволяющих записывать сложные фрагменты программного кода на JavaScript гораздо короче. Например, следующий фрагмент позволяет обойти в цикле значения, находящиеся в массиве, не заботясь об их индексах:

Пример: (исходный файл: `array.coffee`)

```
for name in array
      console.log  name
```

Пример: (исходный файл: `array.js`)

```
(function() {
    var name, _i, _len;
    for (_i = 0, _len = array.length; _i < _len; _i++) {
        name = array[_i];
        console.log(name);
    }
}).call(this);
```

В довесок к синтаксическому сахару, язык CoffeeScript помогает также писать более надежный программный код на JavaScript, следя за видимостью переменных и классов, гарантируя использование соответствующих операторов сравнения, и беря на себя решение многих других рутинных задач, в чем вы убедитесь, читая эту книгу.

Языки CoffeeScript, Ruby и Python часто упоминаются вместе и на то есть свои причины. За основу модели CoffeeScript были взяты краткость и простота синтаксиса этих языков. Благодаря этому CoffeeScript создает «ощущение» более современного языка, чем JavaScript, за основу которого были взяты такие языки, как Java [13] и C++. [14] Как и JavaScript, язык CoffeeScript можно использовать в любом программном окружении. Не имеет никакого значения, на каком языке вы пишете свое приложение, Ruby, Python, PHP, [15] Java или .Net [16]. Программный код, скомпилированный в JavaScript, будет работать со всеми ими.

Поскольку программный код на CoffeeScript компилируется в программный код на JavaScript, сохраняется возможность использовать любые библиотеки на JavaScript. Вы можете использовать jQuery, [17] Zepto, [18] Backbone, [19] Jasmine [20] или любую другую библиотеку, и все они будут работать. Такое не слишком часто говорят о новых языках.

Звучит неплохо, но я слышу, как вы спрашиваете: а есть ли недостатки у CoffeeScript, в сравнении со старым добрым JavaScript? Отличный вопрос. Ответ на него: не так много. Во-первых, несмотря на то, что CoffeeScript предоставляет по-настоящему замечательный способ разработки программного кода на JavaScript, он не дает ничего сверх того, что возможно в JavaScript. Например, я по-прежнему не могу создать JavaScript-версию знаменитого метода *method_missing* в языке Ruby. [21] Самый большой недостаток заключается в том, что вам и членам вашей команды придется учить новый язык. К счастью это легко поправимо. Как вы сами увидите, CoffeeScript чрезвычайно прост в изучении.

Наконец, если по каким-либо причинам выбор CoffeeScript окажется ошибочным для вас или вашего проекта, вы сможете взять сгенерированный программный код на JavaScript и работать с ним. Поэтому в действительности вы ничего не потеряете, если попробуете использовать CoffeeScript в следующем или даже в текущем своем проекте (CoffeeScript и JavaScript очень хорошо уживаются друг с другом).

Кому адресована эта книга?

Эта книга адресована разработчикам на JavaScript со средним и высоким уровнем подготовки. Есть несколько причин, по которым я не могу рекомендовать эту книгу тем, кто не знаком с JavaScript, или тем, кто имеет лишь общие представления о нем.

Во-первых, эта книга не учит программированию на языке JavaScript. Эта книга рассказывает о CoffeeScript. Попутно вы, конечно, узнаете кое-что о JavaScript (и CoffeeScript обладает особым талантом заставлять узнавать больше о JavaScript), но мы не будем погружаться в основы JavaScript и постепенно изучать его.

Пример: что делает следующий фрагмент? (исходный файл: `example.js`)

```
(function() {
    var array, index, _i, _len;
    array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    for (_i = 0, _len = array.length; _i < _len; _i++) {
        index = array[_i];
        console.log(index);
    }
}).call(this);
```

Если вам не удалось понять, что делает этот фрагмент, я рекомендую прервать чтение книги на этом. Не переживайте, я действительно хочу, чтобы вы вернулись и продолжили чтение. Просто я полагаю, что вы получите от книги больше, если будете иметь более близкое знакомство с языком JavaScript. Я буду рассказывать о некоторых основных особенностях языка JavaScript, чтобы проиллюстрировать или помочь лучше понять, что делает тот или иной фрагмент на CoffeeScript. Несмотря на то, что для ясности изложения будут охватываться некоторые основы JavaScript, действительно важно, чтобы вы получили базовые знания о языке JavaScript прежде, чем продолжить чтение. Поэтому, пожалуйста, найдите хорошую книгу о JavaScript (их существует великое множество), прочитайте ее и снова присоединяйтесь ко мне, чтобы стать гуру CoffeeScript.

Тем, кто уже является рок-звездой в JavaScript, я предлагаю поднять уровень игры. Эта книга расскажет вам, как писать ясный, более краткий и надежный программный код на JavaScript, используя вкусности CoffeeScript.

Как читать эту книгу

Я попытался так организовать материал в этой книге, чтобы помочь вам построить фундамент в освоении CoffeeScript. Главы в первой части следует читать по порядку, потому что каждая следующая глава основана на понятиях, изучаемых в предыдущих главах, поэтому, пожалуйста, не перепрыгивайте через главы.

В процессе чтения каждой главы вы заметите несколько обстоятельств.

Во-первых, когда я буду представлять какую-нибудь внешнюю библиотеку, новую идею или понятие, я буду включать ссылку на веб-сайт, где вы сможете получить дополнительную информацию о предмете обсуждения. Я очень хотел бы рассказать вам о многом, например, о языке Ruby, однако в книге недостаточно места для этого. Поэтому, если я буду говорить о чем-то, и вы пожелаете познакомиться с этим поближе, прежде чем продолжить чтение, отправляйтесь по указанной мной ссылке, утолите свою жажду познания и возвращайтесь к книге.

Во-вторых, в каждой главе, я время от времени буду сначала показывать неправильное решение проблемы. После знакомства с неправильным путем, мы исследуем его, чтобы понять, что именно в нем неправильно, а затем найдем правильное решение проблемы.

Отличный пример такого подхода встретится вам в главе 1, «Введение», где рассказывается о различных способах компиляции программного кода на CoffeeScript в программный код на JavaScript.

Иногда в книге вам будут встречаться такие примечания:

> **Совет. Какой-нибудь полезный совет.** Так будут оформляться небольшие советы и рекомендации, которые, на мой взгляд, могут вам пригодиться.

Наконец, на протяжении всей книги я буду представлять примеры программного кода сразу по два-три блока. Сначала будет приводиться пример на языке CoffeeScript, а затем скомпилированная версия (на JavaScript) того же примера. Если в процессе выполнения пример выводит какую-нибудь информацию (на мой взгляд, что-нибудь важное), я буду включать также вывод примера. Ниже показано, как это выглядит:

Пример: (исходный файл: `example.coffee`)

```
array = [1..10]
for index in array
      console.log  index
```

Пример: (исходный файл: `example.js`)

```
(function() {
    var array, index, _i, _len;
    array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    for (_i = 0, _len = array.length; _i < _len; _i++) {
        index = array[_i];
        console.log(index);
    }
}).call(this);
```

Вывод: (исходный файл: `example.coffee`)

```
1
2
3
4
5
6
7
```

```
8
9
10
```

Иногда будут встречаться ошибки, которые мне хотелось бы показать. Например:

Пример: (исходный файл: `oops.coffee`)

```
array = [1..10]
oops! index in array
      console.log  index
```

Вывод: (исходный файл: `oops.coffee`)

```
Error: In content/preface/oops.coffee, Parse error on line 3: Unexpected
↪'UNARY'
    at Object.parseError (/usr/local/lib/node_modules/coffee-script/lib/
↪coffee-script/parser.js:470:11)
    at Object.parse (/usr/local/lib/node_modules/coffee-script/lib/
↪coffee-script/parser.js:546:22)
    at /usr/local/lib/node_modules/coffee-script/lib/coffee-script/
↪coffee-script.js:40:22
    at Object.run (/usr/local/lib/node_modules/coffee-script/lib/
↪coffee-script/coffee-script.js:68:34)
    at /usr/local/lib/node_modules/coffee-script/lib/coffee-script/
↪command.js:135:29
    at /usr/local/lib/node_modules/coffee-script/lib/coffee-script/
↪command.js:110:18
    at [object Object].<anonymous> (fs.js:114:5)
    at [object Object].emit (events.js:64:17)
    at afterRead (fs.js:1081:12)
      at  Object.wrapper  [as  oncomplete]  (fs.js:252:17)
```

Структура книги

С целью помочь вам извлечь максимум пользы из этой книги, я разбил ее на две части.

Часть I: Основы CoffeeScript

Цель первой части – охватить весь язык программирования CoffeeScript сверху донизу. К концу этой части вы должны быть готовы

столкнуться с любыми проектами на CoffeeScript, которые встретятся на вашем пути, включая примеры во второй части книги.

Глава 1, «Введение», рассказывает о различных способах компиляции и запуска программного кода на языке CoffeeScript. Здесь также будет представлена мощная утилита командной строки и одновременно интерактивная срда `coffee`, распространяемая в составе CoffeeScript.

Глава 2, «Основы», начинает исследование отличий между CoffeeScript и JavaScript. Рассказ о синтаксисе, переменных, областях видимости и других элементах языка закладывает мощный фундамент для всех остальных глав в книге.

Глава 3, «Управляющие конструкции», сосредотачивает внимание на наиболее важной части любого языка программирования – управляющих конструкциях, таких как условные операторы `if` и `else`. Здесь также рассказывается о различиях между одними и теми же операторами в CoffeeScript и JavaScript.

Глава 4, «Функции и аргументы», подробно охватывает функции в языке CoffeeScript. Здесь рассказывается об особенностях определения функций, их вызове и дополнительных возможностях, таких как аргументы со значениями по умолчанию и групповые аргументы.

От массивов к объектам, глава 5, «Коллекции и итерации», покажет, как в CoffeeScript можно использовать и изменять коллекции объектов, а также выполнять итерации по ним.

Глава 6, «Классы», завершает первую часть книги описанием поддержки классов в CoffeeScript. Она рассказывает, как определять новые классы и расширять существующие, как переопределять методы суперклассов и многое другое.

Часть II: Практическое применение CoffeeScript

Во второй части демонстрируется использование CoffeeScript на практических примерах. После знакомства с некоторыми сторонами экосистемы, окружающей CoffeeScript, а также после создания полноценного приложения, к концу второй части вы должны будете обладать отточенными навыками владения языком CoffeeScript.

Глава 7, «Инструмент сборки Cake и файлы сборки Cakefile», описывает инструмент Cake, распространяемый в составе CoffeeScript. Этот маленький инструмент можно использовать для создания

сценариев сборки, тестирования и выполнения других операций. Здесь будет рассматриваться все, что может предложить этот инструмент.

Тестирование является важной составляющей разработки программного обеспечения, и в главе 8, «Тестирование с помощью Jasmine», дается короткий обзор одной из наиболее популярных библиотек тестирования для CoffeeScript/JavaScript – Jasmine. В этой главе будет продемонстрировано применение популярного шаблона разработки через тестирование, где при разработке класса калькулятора сначала будут написаны тесты.

Глава 9, «Введение в Node.js», представляет собой краткое введение в управляемый событиями серверный фреймворк Node.js. В этой главе будет показано использование CoffeeScript для создания простого HTTP-сервера, который автоматически компилирует файлы CoffeeScript в файлы JavaScript при запросе их со стороны веб-браузера.

Глава 10, «Пример: список задач, часть 1 (серверная)», описывает создание серверной части приложения списка задач. Основываясь на главе 9, прикладной интерфейс приложения будет конструироваться с применением веб-фреймворка Express.js и компонента Mongoose ORM для MongoDB.

Глава 11, «Пример: список задач, часть 2 (клиент на основе jQuery)», описывает создание клиента прикладного интерфейса списка задач, созданного в главе 10, с применением популярной библиотеки jQuery.

Глава 12, «Пример: список задач, часть 3 (клиент на основе Backbone.js)», описывает создание клиента прикладного интерфейса списка задач, но на этот раз с применением клиентского фреймворка Backbone.js.

Установка CoffeeScript

Я не большой поклонник инструкций по установке в книгах, потому что когда книга попадает на книжную полку, инструкции по установке оказываются устаревшими. Однако некоторые (здесь я подразумеваю издателей книг) полагают, что книги должны содержать раздел с описанием процедуры установки. Поэтому ниже приводятся мои инструкции.

Установить CoffeeScript очень просто. Проще всего выполнить установку – это перейти на сайт http://www.coffeescript.org и отыскать инструкции по установке там.

Я доверяю ответственным за сопровождение в таких проектах, как CoffeeScript и Node [22], – они прекрасные специалисты и не забывают своевременно обновлять инструкции по установке, и вебсайты этих проектов – отличное место, где можно найти эти инструкции.

На момент написания этих строк актуальной версией CoffeeScript была версия 1.2.0. Все примеры в этой книге должны работать в этой версии.

Как запускать примеры

Отыскать и загрузить файлы с исходными текстами примеров для этой книге можно по адресу: https://github.com/markbates/Programming-In-CoffeeScript. Как уже было показано выше, для каждого примера в книге указывается имя файла. Файлы примеров сгруппированы в каталогах по номерам глав.

Если явно не оговаривается иное, примеры должны запускаться в терминале, например:

```
> coffee example.coffee
```

Итак, теперь, когда вы знаете, как запускать примеры после установки CoffeeScript, почему бы нам не встретиться в главе 1 и не приступить к изучению? Встретимся там.

Примечания

1. http://ru.wikipedia.org/wiki/JavaScript.
2. http://ru.wikipedia.org/wiki/Ajax.
3. http://www.prototypejs.org/.
4. http://documentcloud.github.com/backbone/.
5. http://ru.wikipedia.org/wiki/Model-view-controller.
6. http://www.adobe.com/.
7. http://www.apple.com/ios/.
8. http://www.coffeescript.org.
9. http://www.rubyonrails.org.

10. http://www.rubyinside.com/rails-3-1-adopts-coffeescript-jquery-sass-andcontroversy-4669.html.
11. http://ru.wikipedia.org/wiki/Ruby.
12. http://ru.wikipedia.org/wiki/Python.
13. http://ru.wikipedia.org/wiki/Java.
14. http://ru.wikipedia.org/wiki/C%2B%2B.
15. http://ru.wikipedia.org/wiki/PHP.
16. http://ru.wikipedia.org/wiki/.NET_Framework.
17. http://www.jquery.com.
18. https://github.com/madrobby/zepto.
19. http://documentcloud.github.com/backbone.
20. http://pivotal.github.com/jasmine/.
21. http://ruby-doc.org/docs/ProgrammingRuby/html/ref_c_object.html#Object.method_missing.
22. http://nodejs.org.

Часть I. Основы CoffeeScript

В этой первой половине книги рассматривается все, что вы хотели бы узнать и все, что вы должны знать о CoffeeScript. К концу этой части книги вы должны быть готовы начать программировать на CoffeeScript, не испытывать неудобств при использовании сопутствующих инструментов и знать все достоинства и недостатки самого языка.

Изучение мы начнем с самого начала, с таких основ, как компиляция и запуск файлов CoffeeScript. Затем мы перейдем к изучению синтаксиса языка CoffeeScript. После знакомства с синтаксисом будут рассматриваться управляющие конструкции, функции, коллекции и, наконец, классы.

Каждая следующая глава будет основываться на предыдущих главах. Попутно вы познакомитесь со всеми хитростями языка CoffeeScript, позволяющими писать потрясающие JavaScript-приложения. Итак, нам многое предстоит узнать – приступим!

1. Введение

Теперь, прочитав предисловие и установив CoffeeScript, можно ли сказать, что вы действительно приступили к использованию этого языка? В этой главе мы рассмотрим несколько способов компиляции и запуска файлов с программным кодом на языке CoffeeScript.

Я расскажу об удачных и не очень способах компиляции и выполнения программ. В этой главе не будут рассматриваться внутренние механизмы CoffeeScript, однако она ценна уже тем, что в ней вы начнете работать с CoffeeScript. Знание всех достоинств и недостатков инструментов командной строки, распространяемых вместе с CoffeeScript, упростит вашу жизнь, не только как читателя этой книги, но и как разработчика, приступающего к созданию своих первых приложений на CoffeeScript.

Даже если вам уже приходилось сталкиваться с инструментами командной строки CoffeeScript, все равно есть вероятность, что вы узнаете что-то новое в этой главе, поэтому потратьте несколько минут, чтобы прочитать ее, прежде чем перейти к главе 2, «Основы».

Интерактивная среда CoffeeScript

В состав CoffeeScript входит по-настоящему мощная интерактивная среда REPL [1], известная также как интерактивная консоль, позволяющая немедленно приступить к экспериментам с CoffeeScript.

Начать работу с консолью REPL очень просто. Достаточно лишь ввести следующую команду в окне терминала:

```
> coffee
```

После этого должно появиться приглашение к вводу, которое выглядит примерно так:

```
coffee>
```

После появления на экране приглашения `coffee` можно начинать экспериментировать с CoffeeScript.

Консоль REPL можно также запустить командой:

```
> coffee -i
```

или, если вам нравится работать на клавиатуре:

```
> coffee --interactive
```

Начнем с простого примера. Введите в консоли следующую команду:

```
coffee> 2 + 2
```

Вы должны получить следующий ответ:

```
coffee> 4
```

Поздравляю, вы только что написали свой первый программный код на языке CoffeeScript!

Теперь попробуем что-нибудь поинтереснее, что-нибудь более CoffeeScript-овское. Не будем пока задумываться, что делает следующий код (об этом будет рассказываться чуть ниже), а просто запустим его.

Пример: (исходный файл: `repl1.coffee`)

```
a = [1..10]
b = (x * x for x in a)
console.log b
```

Вывод: (исходный файл: `repl1.coffee`)

```
[ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
```

Этот фрагмент кода на CoffeeScript выглядит намного интереснее, не так ли? Если говорить в двух словах, мы создали массив и заполнили его целыми числами от 1 до 10. Затем мы обошли все числа в массиве `a`, умножили каждое из них на само себя и создали второй массив, `b`, содержащий эти новые значения. Здорово, правда? Как уже говорилось выше, позднее я буду рад объяснить, как все

это работает, а пока просто порадуемся всем строкам на JavaScript, которые не пришлось писать. Однако, ради любопытства, ниже приводится эквивалентный код на JavaScript:

Пример: (исходный файл: `repl1.js`)

```
(function() {
    var a, b, x;
    a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    b = (function() {
        var _i, _len, _results;
        _results = [];
        for (_i = 0, _len = a.length; _i < _len; _i++) {
            x = a[_i];
            _results.push(x * x);
        }
        return _results;
    })();

    console.log(b);
}).call(this);
```

Как видите, консоль REPL дает отличную возможность опробовать различные идеи и поэкспериментировать с ними. Однако, не все так прекрасно в стране CoffeeScript REPL. Как уже было показано выше, пробелы в языке CoffeeScript имеют большое значение. В результате возникает проблема с вводом многострочного кода на CoffeeScript в консоль REPL. Эта проблема легко решается с помощью символа \.

Попробуем написать простую функцию `add`, принимающую два числа и возвращающую их сумму. Введите следующий код в консоли REPL, строку за строкой:

Пример: (исходный файл: `repl2.coffee`)

```
add = (x, y)->\
    x + y
console.log add(1, 2)
```

Вывод: (исходный файл: `repl2.coffee`)

```
3
```

Обратите внимание, что в конце первой строки добавлен символ \. Он сообщает интерактивной консоли REPL, что далее следует

еще одна строка, продолжающая это выражение. Важно не забывать добавлять символ \ перед каждой следующей строкой, добавляемой в выражение. Первая же строка, не заканчивающаяся символом \, будет считаться концом выражения и консоль REPL попытается выполнить его.

Также важно отметить, что после символа \ по-прежнему необходимо соблюдать отступы, чтобы компилятор CoffeeScript смог корректно интерпретировать эту строку выражения и поместить ее на соответствующее место.

Наконец, чтобы *завершить работу* с консолью REPL, достаточно просто нажать комбинацию клавиш **Ctrl-C**.

Консоль REPL – мощный инструмент, позволяющий быстро опробовать идеи, но, как мы видели, использовать его становится немного труднее, когда приходится иметь дело с более сложным программным кодом. Далее в этой главе, в разделе «Выполнение файлов CoffeeScript», будет показан способ выполнения файлов на языке CoffeeScript, который лучше подходит для запуска сложного программного кода.

Компиляция в браузере

При разработке веб-приложения рано или поздно наступит момент, когда у вас появится желание вставить некоторый программный код на CoffeeScript непосредственно в HTML-файл [2]. Это вполне возможно, и я покажу, как это делается. Однако хочу предостеречь вас от подобного шага. Во-первых, существуют весьма веские причины, по которым такие методики программирования, как ненавязчивый JavaScript (Unobtrusive JavaScript) [3], приобрели большую популярность в последнее время. Не смотря на всю привлекательность возможности выполнить код на CoffeeScript непосредственно в браузере, в действительности это не самое лучшее решение. Отделяя код на JavaScript от разметки HTML, можно сохранить чистоту кода и обеспечить более изящную деградацию функциональности в окружениях, не поддерживающих JavaScript.

Первое время следование методике ненавязчивого JavaScript может показаться немного утомительным и сложным делом, но потом, привыкнув к ней, вы обнаружите, что она упрощает разработку программного кода, более пригодного для многократного использования, и вообще выглядит более логичной. Используя такие инструменты, как библиотека jQuery, можно дождаться окончания

загрузки страницы и подключить весь необходимый программный код на JavaScript к соответствующим объектам в странице. Однако иногда бывает необходимо предварительно «заполнить насос», если можно так выразиться. Обычно это означает, что требуется вызвать метод `init`, возможно, передав для этого некоторые данные в формате JSON [4]. Я мог бы предложить написать этот небольшой блок кода на чистом JavaScript. Однако для желающих написать его на CoffeeScript существует способ позволить браузеру самому скомпилировать этот код.

Рассмотрим HTML-файл с небольшим фрагментом программного кода на CoffeeScript, встроенным в него.

Пример: (исходный файл: `hello_world.html`)

```
<html>
    <head>
        <title>Hello World</title>
        <script src='http://jashkenas.github.com/coffee-script/
➥extras/coffee-script.js' type='text/javascript'></script>
    </head>
    <body>
        <script type='text/coffeescript'>
            name = prompt "What is your name?"
            alert "Hello, #{name}"
        </script>
    </body>
</html>
```

Браузеры, по крайней мере на момент написания этих строк, не имели встроенной поддержки языка CoffeeScript, поэтому к странице необходимо подключить компилятор. К счастью, команда проекта CoffeeScript разработала такой компилятор. Чтобы подключить его к HTML-странице, необходимо добавить следующую строку в раздел `head` HTML-файла:

```
<script src='http://jashkenas.github.com/coffee-script/extras/coffee-script.js'
type='text/javascript'></script>
```

Разумеется, при желании можно загрузить содержимое файла `coffee-script.js` и хранить его локально в своем проекте.

Единственное, что необходимо сделать во встроенном коде на CoffeeScript, чтобы скомпилировать его, это определить соответствующее значение атрибута `type` в теге `script`, как показано ниже:

```
<script type='text/coffeescript'></script>
```

После загрузки страницы компилятор CoffeeScript, хранящийся в файле `coffee-script.js`, отыщет в HTML-документе все теги `script` с атрибутом `type`, имеющим значение `text/coffeescript`, скомпилирует содержащиеся в них сценарии в эквивалентный код на JavaScript и затем выполнит скомпилированный код.

Предостережение

Теперь, когда вы знаете, как скомпилировать программный код на CoffeeScript, встроенный в HTML-документ, я хочу сделать несколько замечаний. Первое, все, что будет говориться в этой книге об областях видимости, об анонимных функциях-обертках и так далее, в равной степени относится и к компиляции CoffeeScript таким способом. Об этом следует помнить, при разработке подобного программного кода.

Второе и, пожалуй, самое важное – это далеко не самый быстрый способ компиляции сценариев на CoffeeScript. Это означает, что после передачи таких страниц в эксплуатацию, все ваши пользователи будут загружать дополнительный файл, размером 162.26 Кбайт, чтобы скомпилировать код на CoffeeScript. А после загрузки страницы компилятору потребуется некоторое время, чтобы отыскать в странице все теги `text/coffeescript`, скомпилировать их содержимое и затем выполнить полученный код на JavaScript. Не думаю, что это будет радовать пользователей.

После этих предупреждений я надеюсь, что вы выберете правильный путь и будете компилировать программный код на CoffeeScript до передачи своих веб-страниц в эксплуатацию.

Компиляция в командной строке

Несмотря на удобство и простоту компиляции и выполнения в браузере программного кода на языке CoffeeScript, в действительности это не самое лучшее решение. Программный код на CoffeeScript следует компилировать до того, как он попадет в Веб. Также вполне возможно, что вам придется писать приложения для фреймворка Node.js или другие серверные приложения, которые выполняются не в браузере и потому не смогут быть скомпилированы там.

Так как же тогда скомпилировать свой код на CoffeeScript? Отличный вопрос. Можно отыскать множество сторонних библиотек, которые скомпилируют ваши файлы с кодом на CoffeeScript (в код на различных языках и на различных платформах), но важнее понять, как сделать это самостоятельно, чтобы можно было написать свой сценарий компиляции, если это потребуется.

Флаг --compile

Начнем с самого важного флага команды `coffee`, `-c`. Флаг `-c` принимает в качестве аргумента имя файла с программным кодом на языке CoffeeScript и компилирует его в файл с программным кодом на языке JavaScript, сохраняя его в том же каталоге. Именно так я компилировал примеры в этой книге.

Сделайте шаг вперед, создайте файл с именем `hello_world.coffee` и добавьте в него следующий текст:

```
greeting = "Hello, World!"
console.log  greeting
```

Теперь скомпилируйте этот файл:

```
> coffee -c hello_world.coffee
```

Эта команда должна скомпилировать программный код CoffeeScript в исходном файле в программный код на JavaScript и сохранить его в новом файле с именем `hello_world.js` в том же каталоге. Файл `hello_world.js` должен содержать следующий текст:

```
(function() {
    var greeting;
    greeting = "Hello, World!";
    console.log(greeting);
}).call(this);
```

Новый файл `hello_world.js` с программным кодом на JavaScript готов к эксплуатации! Теперь можно выпить чашечку чая.

Интерфейс командной строки CoffeeScript

Мы немного поиграли с консолью REPL и узнали, как компилировать программный код на CoffeeScript с помощью инструмента

командной строки `coffee`, однако команда `coffee` предлагает множество дополнительных и интересных возможностей, на которые стоит взглянуть. Чтобы увидеть полный список возможностей команды `coffee`, введите следующую команду в окне терминала:

```
> coffee --help
```

Вы должны увидеть вывод, как показано ниже:

```
Usage: coffee [options] path/to/script.coffee
  -c, --compile      compile to JavaScript and save as .js files
  -i, --interactive  run an interactive CoffeeScript REPL
  -o, --output       set the directory for compiled JavaScript
  -j, --join         concatenate the scripts before compiling
  -w, --watch        watch scripts for changes, and recompile
  -p, --print        print the compiled JavaScript to stdout
  -l, --lint         pipe the compiled JavaScript through JavaScript Lint
  -s, --stdio        listen for and compile scripts over stdio
  -e, --eval         compile a string from the command line
  -r, --require      require a library before executing your script
  -b, --bare         compile without the top-level function wrapper
  -t, --tokens       print the tokens that the lexer produces
  -n, --nodes        print the parse tree that Jison produces
  --nodejs           pass options through to the "node" binary
  -v, --version      display CoffeeScript version
   -h, --help        display this help message
```

(Перевод:

```
Порядок использования: coffee [ключи] путь/к/файлу/script.coffee
  -c, --compile      скомпилировать в JavaScript и сохранить как файл .js
  -i, --interactive  запустить интерактивную консоль CoffeeScript REPL
  -o, --output       указать каталог для скомпилированных JavaScript-файлов
  -j, --join         объединить сценарии перед компиляцией
  -w, --watch        следить за изменениями и немедленно перекомпилировать
  -p, --print        вывести скомпилированный код JavaScript в устройство
                     стандартного вывода
  -l, --lint         пропустить скомпилированный JavaScript через
                     JavaScript Lint
  -s, --stdio        принять сценарий из устройства стандартного ввода и
                     скомпилировать его
  -e, --eval         скомпилировать строку из командной строки
  -r, --require      загрузить библиотеку перед выполнением сценария
```