<<深入浅出CoffeeScript>>

图书基本信息

书名：<<深入浅出CoffeeScript>>

13位ISBN编号：9787115279743

10位ISBN编号：7115279748

出版时间：2012-5

出版时间：人民邮电出版社

作者：[英] Trevor Burnham

页数：104

译者：寸　志

<<深入浅出CoffeeScript>>

内容概要

作为唯一所有主流浏览器都支持的脚本语言，JavaScript俨然已成为Web开发领域最具号召力的语言，但它的种种语言怪癖以及在各种浏览器间实现不一致等问题也为开发人员所诟病。
2009年底，CoffeeScript横空出世。
它吸收了JavaScript语言的精华，并添加了很多现代语言脚本的特性，很快便得到了大量Web开发人员的青睐。

本书由CoffeeScript重要贡献者Trevor
Burnham操刀，从基础知识入手，全面详尽地介绍了CoffeeScript这门新语言。
通过一个5×5拼字游戏，作者将CoffeeScript各方面的知识融入其中，通俗易懂地讲解了CoffeeScript如何与jQuery等非常流行的类库完美集成，如何游刃有余地结合Socket.IO实现Node.js双通道异步通信。
每章结尾都有精心设计的习题，有助于读者巩固所学的CoffeeScript知识并更上一层楼。

掌握CoffeeScript，Web开发之旅将更轻松、快捷和优雅！

作者简介

作者
Trevor Burnham
全栈式Web框架开发专家，DataBraid创始人，开源拥趸，技术布道师。
其Twitter账号@TrevorBurnham和@CoffeeScript拥有大批追随者。

译者
寸志
2011年7月毕业于同济大学信息安全专业，在校期间，曾荣获微软精英大挑战优胜奖，第三届三叶草软件竞赛三等奖等。
现任大众点评网前端工程师。
热爱互联网，热衷研究新技术，崇尚开源精神，喜欢运动和阅读。
豆瓣社区@island205，博http://island205.com/，Twitter账号@island205，欢迎读者与他交流探讨技术问题。

<<深入浅出CoffeeScript>>

书籍目录

章节摘录

入门指南　如果你读过前言，那么现在应该已经了解了CoffeeScript是什么，它从何而来，以及它为什么是继Herman Miller牌办公椅之后，对程序员来说最棒的东西了。
但是实际上你还没写过一行代码，等不及了是吧？
好，深呼吸下，时候到了。
在本章中，我们将在你的操作系统中安装CoffeeScript，配置好编辑器，最后再运行一些代码！
1.1　安装CoffeeScript　CoffeeScript编译器是用CoffeeScript写成的，这就产生了一个先有鸡还是先有蛋的问题：我们是如何在一个还没装CoffeeScript编译器的系统上运行编译器的呢？
如果能找到某种方法，在机器上浏览器之外运行JavaScript代码，且允许这些代码访问本地文件系统就好了……　对，其实我们有Node.js！
大家把Node当成一个JavaScript的Web服务器（详见6.1节），但是它可不止这个功能。
从根本上讲，它是JavaScript代码和操作系统之间的一个桥梁。
Node也有一个名为npm的很棒的工具，即Node包管理器（Node Package Manager）。
如果你是Ruby程序员，可以将其想象为Node版的RubyGems。
npm已经成为安装管理Node程序和类库约定俗成的标准了。
本节的剩余内容讲述Node和npm的安装，有了它们，我们就能够使用CoffeeScript标准的coffee编译器了（我们在第6章同样需要使用Node和npm）。
如果你迫不及待地想要实践一下的话，可以访问http：//coffeescript.org/，点击“Try CoffeeScript”按钮，然后直接跳到下一章去（要在浏览器中显示console输出，需要某些工具，比如说Fire Lite）。
准备好了？
那我们就开始吧。
使用Node.js和npm安装CoffeeScript　尽管有很多不借助Node来运行CoffeeScript代码的方法（附录2会谈到其中几种），然而我还是假定你在全书中用的是标准的coffee命令，专门运行在Node上的。
但是只有在第6章才会明确需要使用Node和npm。
请注意，使用Windows系统的用户，在继续之前你需要先安装Cygwin。
Cygwin基本上相当于一个Linux模拟器。
虽然Node.js在0.6版本的蓝图中计划直接支持Windows，但是在写作本书之时，使用Cygwin是现有的最可靠的方法。
Mac用户需要安装Xcode，重点并不在于这个程序，而在于那些随它一起安装的命令行开发工具。
尝试运行命令gcc（GNU编译器集合）来检测系统中是否已经安装了这些工具：　如果输出如上所示，那就说明准备就绪了。
如果没有的话，那么就请安装Xcode（Mac用户），或者直接安装标准创建工具（Linux或者Cygwin环境下）。
无论是什么系统（Linux/Unix/Mac），现在都配置好标准创建工具了吧？
太棒了！
现在去访问http：//gist.github.com/579814，此处列出的安装方法之多会让你眼花缭乱，它们都出自npm的创建者Isaac Schlueter。
对于所有Mac用户，我推荐使用Homebrew方法（先安装Homebrew）。
对于其他系统的用户，列表中的第一个选择则最为直接，也是最好的方式。
Node是个很大的程序包，安装它需要花几分钟。
安装好Node之后，运行最新的npm远程安装脚本：　如果你碰到权限错误，可以使用chown改变Node安装目录的属权（该方法可以减少很多麻烦），也可用sudo sh替换普通sh。
无论选择哪种方法，都要测试一下node和npm是否已经存在于系统的环境变量PATH中了：　（简单的提一下与版本相关的事情：Node的版本号为偶数时API保持稳定。
因此，本书的例子在最新的0.4.x版本下应该运行正常。

但是Node 0.5.x版则会以API的变化为重点，而这些变化将会包含到0.6.x稳定版中。
说到npm，本书中假定你使用的是npm 1.x。
因此，如果你还在使用npm 0.x，是时候升级了。
）　现在抓取最新发布的CoffeeScript：　参数-g是--global的缩写，它使已安装好的库在全局系统中都可用（默认情况下，npm install [package]把指定的程序包安装到当前的子目录node_module中，这样便于安装只适用于特定项目的类库）。
只要是安装那些包含二进制可执行程序的程序包，我都推荐使用-g参数。
　npm install命令的输出结果告诉我们，作为安装包的一部分，两个二进制可执行程序cake和coffee已安装好了。
让我们测试下coffee是否已经在系统的PATH 中了：　如果这样不行，那就看一下npm install输出结果中->符号之前的路径（例如/usr/local/bin），然后把它添加到系统的PATH中去。
如果使用的是Mac默认bash终端的话，在你的~/.profile文件中添加下面这行代码即可：　注意不要遗漏：$PATH这部分，否则/usr/local/bin会直接替换掉系统的PATH变量，而不是将自己添加到里面！
要让这行代码生效，需要保存好文件并且开启一个新的会话终端（比方说，把老的终端关掉打开一个新的）。
　如果使用的是其他系统或终端，步骤可能会略有不同，可以输入echo $SHELL搞清楚你使用的是哪个终端。
不要忘了在修改完文件之后重新打开会话终端，以便修改生效。
　最后一步：就像要想在任何地方都能够使用二进制程序就必须把它们放到PATH中一样，npm安装的Node类库也必须添加到NODE_PATH中。
可以输入如下命令查看Node安装类库的位置：　（该命令同时还列出了npm全局安装的所有类库。去掉-g就可以看到安装在当前目录下的所有类库。
）我们需要把该路径下的子目录node_module添加到NODE_PATH中。
在笔者的系统中，就是将如下内容添加到~/.profile文件中：　同样，你的系统上需要采取的操作步骤可能会有所不同。
要测试NODE_PATH是否有效，打开一个新的会话终端输入命令node，即可打开Node.js的REPL ——一个交互式命令运行环境。
接着输入：　我保证，这是本书中唯一一行你需要输入的JavaScript代码！
　如果NODE_PATH设置得不正确，会看到一个Error：Cannot find module ‘coffee-script’的错误提示。
如果只是看到一段很长的对象描述，那就没有问题了。
完成后，可以输入process.exit()或者使用 – 来退出Node的REPL。
　顺便说一下，coffee-script库已经超出了本书的范围；我能说的就是，在CoffeeScript或JavaScript程序中，它能让你把CoffeeScript编译成JavaScript。
你可以基于此做一些非常酷的事情，比方说你可以自己写一个包含自定义后期处理的编译器，或者可以写一个像Cakefile 那样的打包脚本。
　嘿！
我知道安装过程似乎花了很多时间，不过请相信我，既然我们获得了为自己所用的Node和npm的全部能力，那付出终将获得回报。
现在让我们来配置下编辑环境吧。
　在刀锋上起舞　如果你一定要用最新的CoffeeScript，这实际上也非常容易。
只需要使用git 把CoffeeScript的代码仓库克隆下来，然后使用npm从本地目录中安装它即可：　这将安装CoffeeScript当前的master分支，它多少有点不稳定。
可以运行如下命令来还原到特定版本的CoffeeScript（比如说1.1.1）：　1.2　CoffeeScript编辑器
在https：//github.com/jashkenas/coffee-script/wiki/Text-editor-plugins上可以找到一份最新的支持CoffeeScript的编辑器列表。
如果你使用的是Mac系统，那我推荐使用由Jeremy Ashkenas维护的TextMate插件 。

在撰写本书时，Vim、Emacs、gedit、jEdit以及IntelliJ IDEA也分别有插件提供了对CoffeeScript的支持。
　　最近，使用基于Web的编辑器编写代码已成为可能，这些编辑器支持实时协作，不依赖于任何特殊的设备。
目前对CoffeeScript支持得最好的Web编辑器是安装了Cloud9 Live CoffeeScript Extension 的Could9。
　　当然，你可以使用任何自己喜欢的编辑器，但是支持CoffeeScript的编辑器会给你带来3大优势——语法高亮、自动缩进以及内置的编译快捷方式。
前两个优点理解起来很容易，但是第三个优点是很多程序员没有好好利用的部分。
　　在TextMate中，可以使用 R（运行）来运行CoffeeScript文件，或者只用 B（生成）来查看编译后的JavaScript。
编译只需几毫秒，因此如果对于一个CoffeeScript表达式如何转化为JavaScript不是很确定，那么快速编译就是搞清楚这一过程的最快方法。
如果有被选中的文本，则这些命令仅仅运行选中部分的代码而不是整个文件，这就让测试小块代码以及定位语法错误变得容易多了。
如图2所示　图2　直接在TextMate中运行选择的代码　稍微注意下，一些编辑器（包括TextMate）不会默认采用PATH值，这就意味着在你试图运行coffee命令时可能会出现类似于command not found的错误。
如果遇到这种问题，打开编辑器的配置（可能在Shell Variables下面）设置PATH，以匹配在终端中运行echo $PATH命令时得到的输出值。
你愿意的话也可以顺便设置下NODE_PATH。
　　1.3　“邂逅”coffee　既然你已经把编辑器设置好了，那就是时候介绍标准命令行编译器coffee了。
让我们从必修的“Hello world！
”程序开始。
打开编辑器，创建一个名为hello.coffee的文件，添加如下内容：　直接运行它：　有几件事情你可能会感到奇怪：首先，console.log函数是从哪里冒出来的？
（答案：它是一个Node.js的全局函数。
）其次，JavaScript在哪里呢，不是说CoffeeScript会编译为JavaScript吗？
　　事实上coffee会将hello.coffee隐式地编译为JavaScript，然后将输出结果直接传递给Node，以使其立即执行。
如果这不能满足你的需求，可以使用coffee众多选项中的一个或多个，使用coffee -h命令可以查看这些选项：　如果想查看刚才编译器隐藏的JavaScript，可以运行：　可以查看1.3.1节“包裹中的JavaScript”专题对多余两行代码的解释。
　　1.3.1　编译为JavaScript　-c（编译）可能是最常用的参数，它可以把输出的JavaScript保存到文件中。
除了使用.js扩展名代替.coffee之外，新文件的文件名与原始文件的相同。
让我们继续使用咖啡因饮料的主题：　编译输出到相同路径下的一个名为mochaccino.js的文件中。
使用-o（输出）参数并让目标目录名称紧跟其后，就可以把输出保存在其他地方：　该示例读取source目录（包含其子目录）下的所有.coffee并把对应的.js文件写入output。
注意-co是-c-o的缩写。
其顺序很重要：输出目录名必须紧接在-o之后。
　　另外一个比较常用的参数是-w（监听），它可以让coffee命令在后台持续运行。
结合-c，它在每次开发者作出改变之后重新编译代码。
它甚至能在多目录下工作且能保持嵌套的目录文件结构不变。
因此，如果运行下面的命令，coffee目录下的所有文件都会不断地被重新编译到js目录中：　它会持续运行直到使用 – 来终止编译器。
　　包裹中的JavaScript　你可能想知道为什么CoffeeScript编译后的代码会被包裹在一个函数内？
原因用一个词来说就是命名空间。

如果将一堆JavaScript文件上载到一个浏览器程序中，它们会被当做一个大的代码块，这容易产生不可预料的结果：　写第一个文件的人，对代码可能造成的破坏一无所知！
为避免发生灾难可以把每个文件用一个匿名函数包裹起来，这样就隔开了两个declareNuclearWar声明（参见2.2节），这种方式叫做模块模式。
　为了让模块之间可以互相通信，必须“输出”一些变量（我们会在4.1节详细介绍）。
　如果一定要除去包裹函数，使用-b（暴露）参数来运行coffee命令即可。
　1.3.2　REPL　不带任何参数直接运行coffee会进入编程老手所说的REPL，即Read-Eval-Print Loop。
通俗地说，就是你输入点什么，它执行，然后你查看输出结果，周而复始。
　这很适合用来小试一下这门语言。
REPL运行在Node.js环境中，并且它会输出所有表达式的结果。
例如，如果我们想回忆一下JavaScript中parseInt的某些怪异行为，可以这样试试：　coffee相关的内容就介绍到这里。
再顺便说一句，如果想了解coffee是如何工作的，可以查看带注释的源码。
如果你愿意，甚至可以对其进行反向工程，编写自己的CoffeeScript编译器接口（就像笔者写的Jitter 一样）。
　不要忘了coffee只是一个轻量级的工具，它并不提供代码压缩或者编译后自动运行测试之类的功能。
如果想把这些功能添加到自己的项目中，你就应该编写自己的生成脚本，通常就是所谓的Cakefile。
你可以在CoffeeScript wiki 上找到一些Cakefile相关的文档。
　几乎可以编写CoffeeScript代码了——但还有一个问题，如果遇到错误该怎么办呢？
　1.4　调试CoffeeScript　很多使用类似CoffeeScript这类语言编写代码的人都会遇到以下问题，即运行时错误参考的是编译后的代码而不是原始代码。
这确实是个问题，而且大家也探讨过几个解决方案。
可不幸的是，目前留给你的只有那些行号与原始代码没有任何关系的栈跟踪信息。
　幸好，CoffeeScript编译后的JavaScript有很强的可读性。
如果你了解这两种语言之间的对应关系（我希望读完本书后你能做到这一点），那么在原始CoffeeScript代码中找到与程序中出错的地方相匹配的位置就非常容易了。
　虽然不甚理想，但这就是站在技术最前沿所要付出的代价。
随着CoffeeScript生态圈的日渐成熟，工具越变越好，追踪错误将会越来越容易。
Mozilla基金会的那些家伙为了给Firefox添加CoffeeScript调试支持正在拼命工作。
Node也不会落后太远。
但在此之前，还是彻底测试你的代码，使用调试模式日志，搞懂你的JavaScript代码吧。
　有中间选择么？
有的，在装配了开发控制台（或者像之前提到的Firebug Lite之类的书签工具）的Node.js或者浏览器中，可以使用console.log来显示消息。
这可能会有两个问题：一是你并不想要输出每个细节，二是如果console.log不存在的话你就不会调用它。
通常的解决方案就是使用包装函数，但是这样的话，当输出内容时就无法获得关键的JavaScript代码行号（因为所有的日志都是从同一个地方，即包装函数里输出的）。
因此我推荐如下方式：　在这个例子中，当且仅当地址栏的“哈希”中包含字符串debug（比如page.html#debug），并且浏览器中存在console对象时，debugMode才为true。
这为你提供了一种非常容易的方法，确保在页面加载时能够开启或关闭输出所有额外的信息。
将debugMode声明为window的属性可以让其成为全局变量。
　一种更简单但没有那么通用的方式是使用吸收操作符（详见3.1.4节）以保证当console存在时才调用console.log：　在Node下，有大量的类库（可以使用谷歌快速搜索node logging library）能够显示不同冗余度的输出。
我的styout 也包含其中，它还提供对控制台色差输出的支持。

日志信息可以代替注释，在开发过程中它提供了更多关于代码如何运行的信息。
比如，下面是一段典型的注释完好的代码： 可以像下面这样，调用console.log来代替注释： 另外一种习惯是在代码中使用断言，标准的console对象中有一个assert函数对此提供很好的支持，它接受一个值和错误信息作为参数（值为非真时显示错误信息）。
最后，编写结构良好的代码是避免错误的最重要的保证。
尽管现在还不存在任何工具可以指出导致运行时错误的确切代码行号，但至少应该能够查到程序中可能引发错误的那部分代码。
1.5 预备 本章中我们学习了如何使用Node.js和npm在你的机器上安装CoffeeScript。
你还使用自己最爱的编辑器与这门语言来了次亲密接触，探究了使用CoffeeScript作为开发流程一部分的一些方法，并且认识到了调试工作的挑战性。
既然现在你已经知道了如何运行CoffeeScript代码，是时候深入了解该语言自身的具体细节了。
本书的剩余部分会有大量小代码段，跟上思路的最好方法就是在编辑器中运行这些代码。
如果搞不清楚它们是如何工作的，尝试修改一两行代码看看会发生什么。
你也可以时不时地看一下编译后的JavaScript代码。
想要运行下面这种涉及某个文件的代码段，还需要额外的代码段才行：
GettingStarted/outOfContext.coffee 那些并不涉及某个文件的代码段则能独立运行： 请相信我，如果你的编辑器配置了运行命令的话将更加有趣，只需轻轻敲击快捷键就能查看代码运行结果了。
这是CoffeeScript初学者的最好伙伴！
……

媒体关注与评论

“学习CoffeeScript有助于读者成为更优秀的JavaScript开发人员。
而且，本书读来酣畅淋漓，对于准备学习CoffeeScript的新手，这种体会尤为深刻。
” ——Brendan Eich，JavaScript之父 “很难想象现如今会有哪个Web程序没有大量使用JavaScript。
如果你用惯了Ruby之类的语言，再使用JavaScript就会明显感觉在退步，这可不是什么愉快的事儿。
来看看CoffeeScript吧：它是一个JavaScript预编译器，移除了JavaScript中不必要的冗余，让代码编写和源码阅读变成一件乐事。
来，向着Coffee前进吧！
这是一本很棒的CoffeeScript入门书。
” ——David Heinemeier Hansson，Ruby on Rails之父 “CoffeeScript是编程语言领域最有意思的进展之一，它吸纳Ruby和Python等语言之精华，是一个极富表现力的语言。
本书将指引你进入CoffeeScript的世界；对于那些有志于提高JavaScript开发效率的开发者，本书同样必不可少！
” ——Travis Swicegood，《版本控制之道——使用Git》作者

<<深入浅出CoffeeScript>>

版权说明