Ruby 和 Rails：朴实而深远的朋友

作者 David A. Black译者 Jason Lai 发布于 2007年10月18日 下午7时30分

社区 Ruby 主题 Web框架 交付价值

People talk a lot -- but separately -- about Ruby and Rails having low barriers to entry. "Ruby gets out of your way," I've been hearing over the years, "so you can concentrate on your goals." And Rails gets described frequently in similar terms.

许多人都说Ruby和Rails的入门门槛很低——而且大家都是不约而同地表达了同样的观点。“Ruby语言帮你解决了很多事情，”这些年我听到的都是这样的话语，“因此你可以专注于实现你的目标。”而你也常常会听见人们用相似的好评褒扬Rails。

I happen to agree, in both cases.

对于这两种情况的说法，我恰好都表示赞同。

Of course this doesn't mean that someone chosen at random off the street can write working Ruby code or get a Rails application up and running. Low is relative; you have to be pretty comfortable already with the programming process to reap the rewards of transparent programming technologies. You have to have a way, before Ruby or Rails can get out of it.

当然，这并不是说你从街边随便揪出一个人就可以写出能正常工作的代码或者让一个Rails应用程序启动并正常运转。门槛低是相对的；你得先对习惯于程序开发的过程才能体会到透明编程技术所带来的好处。你得先入道，才能借助Ruby或者Rails在原有的“道”上取得突破。

Ease of use is also relative to the user. As we know from a couple of decades of Usenet and other on-line discourse, there is no such thing as consensus on programming tools. It's nauseating to think of all the rhetoric wasted - yes, wasted, every syllable of it - on the non-existent process of people convincing each other to switch tool sets. (I've been reading Usenet and other forums since 1990. I have never seen anyone convince anyone to switch to anything.) So there's no point bothering to claim that a given language or framework is actually more transparent or better organized or easier to use than another.

对于用户来说，易用性同样也是相对的。数十年来，从Usenet和其它在线评论中我们可以发现，人们对于编程工具一直没有一致认可的选择。所有这些希望说服对方改变自己所用的工具集合的高谈阔论都是在浪费时间——没错，浪费时间，这些话里面的每一个字都在浪费时间——想想都令人作呕。（我从1990年就开始关注Usenet和其它在线论坛了，但从没看见有谁成功说服另外一个人转投某个语言或者工具。）因此，费尽心机宣称某个语言或者框架比起另一个更透明或者组织得更为良好，或者更容易使用，这根本没有意义。

But when you hear the same technologies acclaimed in the same terms over and over again, you have at least got something to remark on and talk about.

不过，要是你听见某些技术被人们以同样的语气被不停赞扬着的话，那么你至少得好好

留意并且讨论一下这些技术了。

In theory, at least, programming languages and systems like Ruby and Rails are instrumental; that is, like a musical instrument, they are a means to an end, a tool for accomplishing a goal, not a goal in themselves. However, just as musicians identify with the history, culture, and technical peculiarities of their instruments, so too do programmers identify with their languages and systems. Yes, you can describe a programming task algorithmically, and then implement it in any of hundreds of languages - just as you can play a given tune on any of hundreds of instruments. But that's not how it ends up happening.

至少在理论上像Ruby和Rails这样的编程语言和系统都是为了帮助你解决某方面具体问题的，也就是说，像乐器一样，它们是让你达到结果和完成目标的工具，而它们本身并不是目的。不过，正如音乐家把自己乐器的历史、文化和技术特性视为一个整体，程序员们对于自己所用的语言和系统也存在同样的认同感。没错，你用算法就可以描述出一个编程任务，然后可以任选几百种编程语言中的一种来把任务实现出来——正如你可以使用数百种乐器中的任意一种演奏出一个特定的曲调是一个道理。不过使用了哪种乐器并不是最终的结局。

There are lots of interesting analogies between being a musician and being a programmer. Some of them, moreover, have to do with the question at hand, the question of ease of access.

将音乐家和程序员做类比的例子简直是不胜枚举，而其中有一些类比，与手头的问题——易于入门的问题，是很有关系的。

Violinist Itzhak Perlman has described the difference between the violin and the piano in these terms - not that one is easy and the other difficult, over the long haul, but specifically that the piano, unlike the violin, gets out of your way. Says Perlman:

小提琴大师Itzhak Perlman（伊扎克·帕尔曼）就恰如其分地点评了小提琴和钢琴的区别——从长远角度看，两者没有哪个更难也没有哪个更简单，但具体说起来，和小提琴不一样，钢琴上手还是更容易一些。Perlman如是说：

> Violinists have a harder time to make pure music than pianists, because pianists ... are immediately forced to turn the phrase. They don't have to deal with vibrato, they don't have to deal with shifting, they don't have to deal with sliding, they don't have to deal with bow-speed .... [On the piano,] basically you put down the key and you get a sound.... You have to deal with music immediately.
>
> 和钢琴演奏者相比，小提琴演奏者要度过一段更为艰苦的历程才能演奏出真正的音乐，因为钢琴演奏者……一上手就得照着乐谱练曲子。他们不用处理颤音（vibrato），不用操心换把（shifting），不用对付滑音（sliding），也无须顾虑弓速（bow-speed）……你只要[在钢琴上]摁下一个键就能听到声音……你一开始就得把音乐弹出来。

In this light, the piano emerges as the Ruby, or the Rails, of the musical instrument kingdom

正因如此，钢琴就成了乐器王国中的Ruby或者Rails一样脱颖而出。

Perlman characterizes the low entry barrier as a responsibility: pianists can't postpone their responsibility to produce beautiful music, because they don't have the excuse of mechanical difficulty with the instrument. (That's not to say it's easy to play the piano well, but only that you can literally drop a book on a piano and get something. You can't do that with a violin.)

Perlman把低入门门槛描述成一种责任：钢琴演奏者无法推脱自己演奏动人音乐的责任，因为他们无法以钢琴在器械操作上有难度为借口。（我们当然不是说你轻而易举就可以把钢琴弹得行云流水，而只是说在你面前扔一本乐谱你就能弹出一点什么，而初用小提琴你是做不到这样的。）

Bragging rights - not the Perlman brags, but he leaves the question playfully hanging - belong to the musicians who play the high-barrier instruments. They produce beautiful phrases against much higher odds. In the computer world, however, the bragging rights go to those whose tools do get out of their way. And Ruby and Rails both come in for their share of getting bragged about.

于是，显摆——当然不是说Perlman在自夸，不过他开玩笑地把这个未决的问题抛给了更多的人——就成了演奏高难度乐器的音乐家专属的权利了，这些人能用“奇技淫巧”演奏出悠扬的乐章。在计算机世界中，显摆的权利则属于使用能让自己效率更高的工具的人们，而Ruby和Rails则成为人们“吹牛显摆”所用的扩音大喇叭。

But the separate discourses about Ruby and Rails, and their approachability is not as remarkable as a third term in the equation - a counterpoint to the main themes, so to speak. And that's the often-expressed doubt or concern about whether or not Rails developers should take the time to master Ruby.

但是，我们应该分开讨论Ruby和Rails，它们入手的难易度可不能直接就划上等号——这也是和我们文章主基调相对应的。这个说法也代表人们常会表达的疑虑：到底Rails开发人员该不该花时间精通Ruby呢？

The question is understandable, from one perspective. The fact that both Rails and Ruby have the "easy access" reputation doesn't mean that they're both easy for the same people, or in exactly the same ways. System S doesn't have to have the same difficulty level as Language L, just because it's written in Language L. Ruby, for example, is written in C, and whatever people say about C, it's usually not that it's easy to get started with.

从某个角度讲，这个问题是可以理解的。Rails和Ruby“门槛低”的美名并不意味着它们对于同一群人来说都很容易，或者说不意味着它们容易的方式是相同的。S系统并不因为使用L语言写的，就一定和L语言具备相同的难度。比如说，Ruby是使用C语言写的，不管人们对C语言如何褒扬贬抑，要说得上C语言入门通常是没那么简单的。

I've seen the case made against mastering Ruby on two separate grounds:

- Rails is easy, but Ruby is much more advanced (too high level, in the sense of difficult).
- Rails is the system; Ruby is the raw material (too low level, not in the sense of easy).

对熟练掌握Ruby语言的反驳，我见过两种情况，分别基于不同的立场：

- Rails 很简单，但 Ruby 就高深得多了（从难度的意义上说，级别太高）。
- Rails 是一个系统；而 Ruby 则是原材料（不从简单的意义上讲，级别过低）。

The thing about both of these positions is that either of them could have been right - and may indeed be right-, with respect to some future Ruby framework or library yet unborn. But neither of them correctly describes the relationship between Ruby and Rails.

对于这两种立场，就未来的Ruby框架或者还没有问世的类库来说，任意一种都可以说是正确的——而且也可能确实是正确的，但不管哪一个都没能正确描述Ruby和Rails之间的

关系。

The key here is the way Rails is engineered. Rails invites to the use of the resources of the Ruby language by developers. Yes, Rails is in many ways a system in itself. But in many, many other ways, Rails exposes, explores, and exploits its connections to Ruby, rather than hiding or disguising them.

这里的关键点在于Rails的设计方式。Rails让开发人员用到Ruby语言中的许多资源。是的，在许多方面Rails都是自成一套系统的。然而在许许多多其它的方面，Rails暴露、探索并且发掘了它和Ruby的联系，而不是将这些联系隐藏或者掩盖起来。

Take one example: helper files. A helper file gets created for you every time you generate a controller. The helper file is empty: it's a place where you can put Ruby methods that you write and that you want to call from your view templates.

举一个例子：helper文件。在你每生成一个控制器的时候helper文件也会相应被创建出来。Helper文件里面什么也没有：你可以在里面放入自己写的Ruby方法，并可以从视图模板中调用这些方法。

There's no abstracting away from Ruby here, no encapsulation of Ruby into a separate domain-specific language, no Wizards Only sign on the door. There's just space - space provided by Rails for your custom Ruby code, space provided so that you can add Ruby code to your application.

这里并没有什么和Ruby无关的抽象，也没有将Ruby封装成一个单独的领域特定语言，更没有“只能通过向导生成（Wizard Only）”的标志。有的只是空位，由Rails提供的空位，可以让你放入定制的Ruby代码，也就是提供给你让你可以为应用程序加入Ruby代码的空位。

Rails encourages you to use Ruby, and to know Ruby. Ruby is neither too hard to use nor too low-level to be of direct, immediate help in your Rails work.

Rails鼓励你使用Ruby，鼓励你去了解Ruby。Ruby既不会过于难以使用，也不会过于底层，而使得你无法给Rails项目带来直接迅速的帮助。

Are Ruby and Rails equal partners, in a simple side-by-side relationship? No; it's more complex than that. Ruby is the antecedent and foundational technology; and Rails does, in some respects, present you with an altered, purpose-driven Ruby environment.

Ruby和Rails是不是两个平级的搭档，二者之间只是一个平行的关系？答案是否定的，其关系要更加复杂一些。Ruby是一项先行的基础技术；而，Rails在某些层面上则为你提供一个经过量体裁衣的、以目的驱动的Ruby环境。

And yet - when you're writing a Rails application, Ruby really does feel like a kind of partner technology. Think of the relationship between XML and XHTML (or any other XML application). One is a system; the other is a system written using the first system. Yet they're both called "markup languages". The naming scheme collapses a level of indirection.

尽管如此，在你编写Rails应用的时候，Ruby看起来确实像一个辅助技术。试想一下XML和XHTML（或者任何其它的XML应用）之间的关系吧。一个是系统，另外一个则是使用第

一个系统写出来的系统。尽管二者都被称为“标记语言（markup languages）”，但是它们的命名方案将其中的一个间接层（indirection）给隐藏起来了。

With Ruby and Rails, it's sort of the opposite. The names are distinct, but the technical relationship is very egalitarian.

对于Ruby和Rails来说，事情有点儿倒过来了。名字是不一样，但是技术上的关系倒是比较平级（egalitarian）。

Or maybe egalitarian isn't exactly the right word. Maybe there is no single right word to describe the Ruby/Rails ratio. Perhaps it's a case for duck typing, the principle that it doesn't really matter what an object is, in terms of class and module, but only what it does. When you're developing a Rails application and you decide you need to write a Ruby method to help you do it, it really doesn't matter whether or not you have a word for the relationship between language and framework. You just reach for what works: Rails facilities, Ruby core classes, plugins, Ruby libraries, methods you write yourself....

可能“平级”这个词用得并不是很准确，可能根本不存在描述Ruby和Rails之间比值的正确用词。可能这就是duck typing的一个例子，其所代表的原则是，无论对象属于什么类或者模块，关系都不大，关键在于它做些什么。当你开发一个Rails程序，并且你决定要写一个Ruby方法来帮你解决问题的时候，你是否能给语言和框架的关系下个定义其实并不重要。你要的只是能给出结果的东西：Rails的设施、Ruby的核心类、插件、Ruby类库、你自己写的方法，等等。

It all flows into the same stream.

这也正和“百川汇于海”是一个道理。

关于作者

David A. Black (dblack@wobblini.net) is the author of Ruby for Rails, from Manning Publications, and a co-director of Ruby Central, Inc., the parent organization of "RubyConf" and "RailsConf", the annual International Ruby and Rails conferences. David provides Ruby and Rails consultancy and training through Ruby Power and Light, LLC (http://www.rubypowerandlight.com).

David A. Black（dblack@wobblini.net） 是Manning出版社《Ruby for Rails》一书的作者，同时是Ruby Central, Inc.的共同主管。Ruby Central, Inc.是RubyConf和RailsConf这两个国际Ruby及Rails年度大会的主办公司。David通过Ruby Power and Light, LLC提供Ruby和Rails的咨询和培训（http://www.rubypowerandlight.com）。

查看英文原文：Ruby and Rails: In your face... but out of your way