Rails / Django比较

Ben Askins and Alan Green

Ben Askins和Alan Green本文最初在2006年12月5日至8日在澳大利亚墨尔本的开源开发者大会上发布。当时是经过审查，被选为本刊作为优秀标准。

摘要

Ruby on Rails（“Rails”）是Ruby的主要Web编程框架，即使在Ruby社区之外，也被认为是最新一代高性能，开源Web开发工具的缩影。 Django是许多竞争的Python开发框架之一。值得注意的是，首先是在Python程序员中受到高度重视，其次是作为Ruby on Rails的新一代框架之一。与传统的Web开发框架相比，Rails和Django都大大提高了生产力。

在本文中，我们从开发人员的角度来比较两个框架，尝试为新项目选择两个框架之一。

1 引言

Ruby on Rails（“Rails”）是Ruby的主要Web编程框架，即使在Ruby社区之外，也被认为是最新一代高性能，开源Web开发工具的缩影。 Django是许多竞争的Python开发框架之一。值得注意的是，首先是在Python程序员中受到高度重视，其次是作为Ruby on Rails的新一代框架之一。

本文希望为这两个框架选择自己开发的开发者编写的。为了比较的目的，作者为他们实施了两次的小型Web应用程序撰写了一个规范：Ben在Rails中有一次，一次由Alan在Django中。然后将这些实施方式通过代码大小和实施时间进行定量比较，以及更多的定性措施，例如检查HTML模板语言的“风味”。

当然还有其他领域也可能与开发人员在Rails和Django之间进行选择，这些都没有在本文中进行研究。这包括相对和绝对的性能，在企业生产环境中部署的便利性以及第三方应用程序和组件的可用性。

Django和Rails应用程序的源代码可以在http://3columns.net/habitual/的subversion版本库中获得。

1.1规格概述

作者实施的小型应用程序被命名为“习惯读者”，它的目的是作为阅读俱乐部的公共网站，让观众看到哪些俱乐部成员已经看过哪些书籍以及他们对每本阅读书的评论。该应用程序还使用标签对书籍进行分类，并从Amazon [http://www.amazon.com/ -Ed]检索信息。除了规范，作者还开发了每个应用程序的七个公共页面的静态HTML原型。基本的HTML和CSS布局是由Andreas Viklund [VIKLUND]修改的模板版本。

习惯读者规范要求应用程序允许记录的读者添加，更改和删除书籍，读者，阅读场合（即读者阅读书籍的场合）和标签。它没有指定这些“管理员”页面的外观和感觉。

1.2习惯读者实现

我们在自己的个人电脑上进行了兼职培训，记录了每个开发任务的时间。我们每个都使用最新版本的每个Web框架 - “Edge Rails”for Ruby和“SVN trunk”for Django - 截至2006年8月。

为了实现这个应用程序，Ben和Alan在Rails和Django方面有着相当的经验。 Ben一直在Ruby和Ruby on Rails上工作六个月。 Ben自九十年代初以来一直在各种数据库支持的软件环境中开发。 Alan只有两个月的兼职Django经验，尽管他自2003年以来一直在开发各种Python框架中的Web应用程序，而从1997年开始就以其他语言开发Web应用程序。

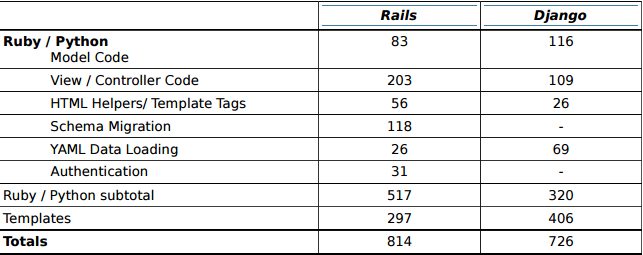
在实施结束时，我们注意到以下规范的变化：1 Django应用程序在每个URL的末尾包含一个额外的“/”。在不影响Django管理员应用程序的情况下，无法轻易更改此行为。

2定量比较

我们使用两个方便的措施来比较两个应用实现：代码行和实现时间。然后使用这些量化措施来通知随附的定性比较的分析。

2.1代码行

在产生这些计数时，我们包括了所有包含大量手写代码的文件。 Rails和Django以其母语（Ruby或Python）生成配置文件，开发人员需要修改它们。我们没有包括这些配置我们没有包含这些配置文件。 Python和Ruby行代码使用David A Wheeler的“scountcount”工具[WHEELER]进行了测量，忽略了空白行和注释。 HTML行计数是通过在每个实现的模板文件上运行“wc”shell命令生成的。

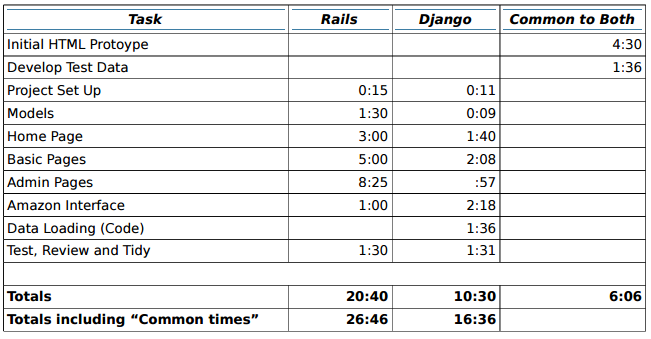


比较主要实现语言，Django应用程序包含比Rails应用程序少三分之一的代码行。如果我们没有在Django实现中实现YAML数据加载，则差异将会更大，因为YAML数据加载计数为Django应用程序中20％的代码行。 （YAML是“另一个标记语言”，它具有方便指定测试数据的语法[YAML]）。

模板代码行的数量也有很大差异。 Django模板的Rails模板与Rails模板一样多，尽管Rails模板包含管理页面，而Django tamplates没有。

2.2实施时间

作者记录了实施习惯读者申请所用的时间，结果如下表所示。我们在三列中记录了时间：在Rails实施上花费的时间，在Django实施上花费的时间，以及对这两个项目有利的任务的时间花费。所有时间都以经过的小时和分钟计算。



Alan在Django中实现这个应用程序显然更快，而不是Ben在Rails中实现这个应用程序。这个差异的大部分原因可归咎于在Rails中执行大约七个半小时的管理功能所需的额外工作。排除管理页面，并允许诸如工作环境的变化和每个开发人员的经验等因素，实施时间大致相等。

3定性比较

本节比较了Rails和Django的一些属性。

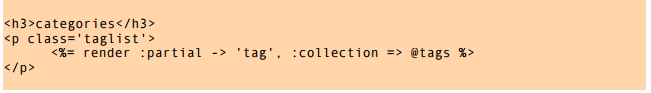
3.1 HTML

模板是Rails和Django的核心功能。两者都允许模板由定义页面的整体布局的基本模板以及定义页面的特定内容的各个页面模板组成。在Rails中，基本模板称为“布局”，而单个页面模板是“视图”。在Django中，都是纯模板。

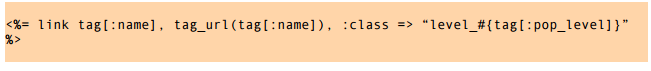
框架之间的关键区别是嵌入动态内容的方式。 Rails视图使用在线Ruby代码片段，因此可能包含任意复杂的功能。Rails鼓励开发人员在他们的视图代码中像他们的控制器或模型代码一样精通，以令人满意的简洁结果回报积极的重构。相比之下，Django使用了一种更简单的模板化语言，特别设计了一些具有HTML技能，但是编程知识最少的网页设计师可以构建模板。例如，在习惯读者的主页上，有一个标签云 - 标签的链接列表，取决于标签的受欢迎程度。 Rails应用程序从数据库检索标签列表，并使其可用于视图。渲染这些标签的视图部分是：



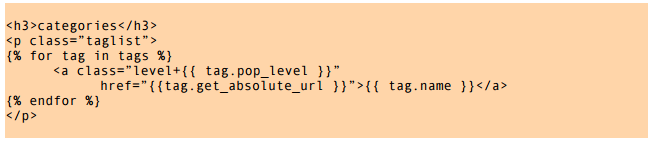
这是指一个Ruby部分，一个页面的片段，它包含在\_tag\_cloud.html中。该文件的内容是：



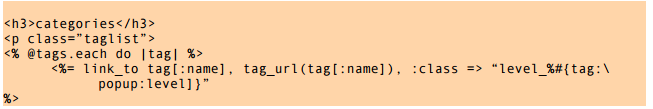
这将生成标题，然后引用另一个部分，它将在名为标签的集合中的每个对象使用一次。 \_tag.rhtml文件包含一行：



这调用一个Ruby助手函数来生成一个链接到标签，适当地设置CSS类。相比之下，生成输出的Django模板的部分是：



Rails视图看起来比Django模板复杂得多，扩展了HTML模板，以在三个独立的文件中生成标签云。 Rails方法的优点是，它允许开发人员重用即使是最小的代码段，从而导致显着更少的重复和更少的模板代码的整体行，即使在诸如习惯读者等小应用程序。但是，Rails不强制这种方法，但是，允许开发人员生成与Django模板几乎相同的视图，如以下示例所示：



Django所采用的更为明确的方法被认为是更简单的教给网页设计师，而没有任何软件开发背景[CROFT]。

3.2模式生成和演化

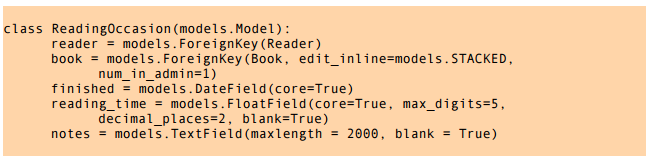
模型对象是Rails和Django中的一个中心概念，通过它们在实现中采用不同的方法。 Rails还为模式演进提供了Django没有的设施。

3.2.1定义模型类

Rails实现了活动记录模式[FOWLER，M]作为其数据访问层。要定义一个模型，开发人员从ActiveRecord基类派生一个新类。 ActiveRecord从数据库中匹配该类名复数的表中推导出模型的属性。开发人员可以修改此映射，但通常认为Rails实践将此类修改保持在最低限度。

另一方面，Django需要开发人员明确地指定每个类，而且每个属性。 Django具有从应用程序模型定义创建数据库模式的标准工具。

来自习惯读者应用程序的模型类作为例子。 ReadingOccasion课程的实例记录读者读书时的场合。 Django代码显式地定义了模型的每个属性，包括定义底层数据库表所需的数据。



Django模型还包含少量用于在自动生成的管理员中指定其外观的元数据页面，如下所述。

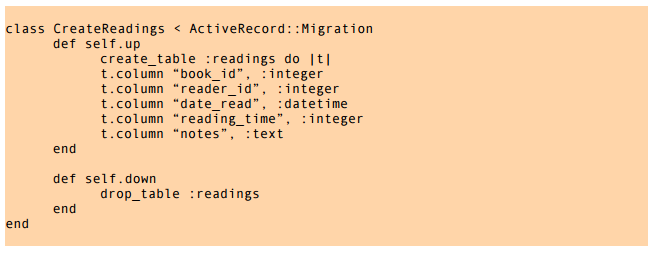
相比之下，Rails版本是最小的。它仅定义自身与其他模型之间的关系。当Rails在运行时检查表定义时，模型的所有属性（读取书籍，读取时间以及其他注释）都会添加：



比较两者，我们看到Rails类的时间要短得多，因为模型的属性没有记录在类定义中。另一方面，Django要求开发人员定义整个模型，包括其属性定义中。这是另一个例子，Rails选择了明确的简洁，而Django选择了相反的。

3.2.2演进模型

Django具有从模型定义创建数据库模式的功能，但并不能为开发人员提供修改（或“演进”）模型定义，同时保留底层数据库表中的数据的任何支持。 Rails迁移机制在创建和演进模型类和基础方案的同时保留数据。迁移是一个Ruby脚本，用于定义对数据库模式的添加，修改和删除。或者，开发人员还可以指定如何将数据从旧版本的模式迁移到新版本。为每个迁移脚本分配了一个版本号，并且可以通过依次应用和反转迁移版本来向后滚动转换模式。这是Rails习惯读者应用程序中用于创建阅读表的迁移：



读取结束调用up方法来应用迁移。它将创建一个名为读数的表格，其中包含给定的列。调用down方法来反转迁移 - 在这种情况下删除读数表。在了解迁移时，重要的是要注意，迁移脚本的唯一影响是在脚本运行时修改模式。当Rails应用程序正在执行时，模型类读取数据库模式以在运行时动态确定其属性。

与Django相比，Rails的增量迁移有两个关键优势。首先，Rails提供了一种标准机制，用于将新版本部署到已经运行的生产系统，同时保留数据。例如，如果数据库列的类型从char更改为整数，则随附的Rails迁移脚本将指定将数据从旧字符列移动到新的整数列所需的步骤。要在Django中执行类似的操作，开发人员需要编写一个特殊的SQL脚本。

第二个优点是容易回滚，迁移鼓励使用模型类和数据库模式进行一定量的实验。当然，在Django中可以对模型进行一些实验，特别是如果模型代码保持在源代码控制下。但是，由于数据不能通过这种更改进行保留，因此除非有快速加载测试数据的机制，否则数据不太有吸引力。

在撰写本文时，Django开发社区正在努力引入模式演进机制。

3.3自动生成的管理页面

许多Web应用程序都有一组“管理员”页面，少数受信任的用户使用的页面，可能会输入内容以发布到更广泛的受众，或维护参考表。

Rails和Django之间有一个明显的区别，就是Django自动生成的管理页面，可以节省大量的开发时间。例如，在习惯读者实施中，Rails中的管理页面的开发占用了开发时间的29％，而Django则为6％。

当我们写道Django管理页面是“自动生成的”时，我们的意思是Django只需要开发人员的小提示即可生成它们。开发人员通过向Django模型类及其字段添加属性和参数来提供这些提示。尽管有点费心，这个过程是相当快速的，生成的页面适用于广泛的情况。为了进一步定制外观，开发人员可以为Django提供备用模板。

Rails根据应用程序基础架构和应用程序功能之间的区别选择不实现管理界面。有一些Rails插件旨在满足与Django管理应用程序相同的需求。两个这样的插件是AutoAdmin [AUTOADMIN]和Streamlined [STREAMLINED]（在开发的早期阶段）。建议在自己的项目中使用Rails的开发人员在开始需要管理界面的项目之前检查这些插件的状态。

3.4国际化

虽然我们没有通过习惯读者申请来解决国际化，但这是Rails和Django社区的一个有据可查的问题。

在其当前版本中，一些Ruby核心库（例如String）不是unicode感知的这意味着使用非西方文本时，字符串操作（如长度和分割）将无法正常运行。 Rails核心团队通过将multibyte\_for\_rails插件与Rails ActiveSupport模块合并为ActiveSupport :: Multibyte [RAILSMULTIBYTE]来解决此问题。其结果是，从版本1.2开始，Rails框架将扩展有问题的Ruby核心库，使其具有unicode感知能力。

自2000年发布的Python 1.6以来，Python已经有Unicode支持和GNU gettext [GETTEXT]风格的国际化。Django框架基于这样的标准化机制，用于国际化Django应用程序和针对各个HTTP请求的本地化响应。

3.5集成第三方代码

Rails和Django都可以利用两种扩展的优点。第一个是Rails调用“plugins”，Django调用“应用程序”。这种扩展意识到它正在其各自的框架内运行。第二个是使用Ruby或Python编写的第三方库，但并不是专门开发用于任何一个框架。

Rails插件是可以包含在Rails应用程序中的离散的代码集合。习惯读者Rails实现利用了acts\_as\_taggable插件来支持标记书籍，以及acts\_as\_attachment插件，以支持上传读者图像并将其调整到可管理的缩略图，以在应用程序中使用。在撰写本文时，可以从http://www.agilewebdevelopment.com/下载419个Rails插件。

Django应用程序可以以类似于Rails插件的方式使用。 Django站点可能由几个应用程序组成，每个应用程序可能包含模型代码，模板，模板标签和纯Python库。通常，一个应用程序将包含站点的主要功能，其他应用程序提供诸如注释和管理页面等功能。在撰写本文时，没有可重复使用的第三方“Python论文”，第2卷，第2期，第51期的中央存储库。

Rails和Django可以自由地与各种本机库集成，也就是分别以Ruby和Python编写的库。例如，在习惯读者应用程序中，Rails和Django都通过本地第三方库与Amazon进行通信。

3.6 AJAX

尽管习惯读者应用程序的Rails和Django实现都没有利用AJAX，但它在Web应用程序中变得非常普遍，这里值得一提。

Rails包含许多帮助函数，可帮助发送和响应XmlHttpRequests。使用RJX，Rails开发人员可以编写一个模板，通过生成将在响应对象中发送并在浏览器中执行的Javascript来响应请求。 Prototype和Scriptaculous帮助器方法是可用的，允许这些Javascript库中的函数在RJS模板中使用。这些JavaScript库与Rails的封装并不排除开发人员选择使用JavaScript库。

相比之下，Django只包含一个JSON模块，留下Javascript代码，并为开发人员选择一个Javascript库（如果有的话）。 Django核心团队认为这是Django的优势。然而，其他Django开发人员表示，他们更倾向于通过正式批准的JavaScript库和相关帮助函数的Rails方法。

3.7其他注意事项

此外，开发人员在两个框架之间进行选择可能会涉及以下非技术领域。 3.7.1成熟度

这两个框架都是从2003-2004年期间开发的Web应用程序中提取出来的。 Rails于2004年7月发布给公众，并于2005年7月发布了Django。因此，Rails在社区对框架的贡献方面取得了长足的进展，并于2005年12月达到了里程碑式的1.0版本。目前的Django版本为0.95，并且在达到1.0里程碑版本之前，Django API可能会进一步更改[DJANGOAPI]。目前有12个核心团队成员在Rails存储库[RAILSCORE]上有承诺权，在Django存储库[DJANGOCORE]上有4个权限。 3.7.2市场地位和炒作

在职位上的搜索JobServe [JOBSERVE]和Seek [SEEK]显示Rails的工作需求比Django以6：1的比例更频繁，而Python的出现比Ruby更频繁比例约为4：1。

虽然Python语言的需求比Ruby高，但Rails框架在市场上比Django更为牢固。与Java和J2EE相比，Rails和Django在市场上获得认可时都很年轻。

可以使用诸如Google Battle [GOOGLEBATTLE]，Ice Rocket [ICEROCKET]和Technorati [TECHNORATI]这样的工具进行类似的比较，其中Rails一直在炒作风险中排名第一。

3.7.3工具和实用程序

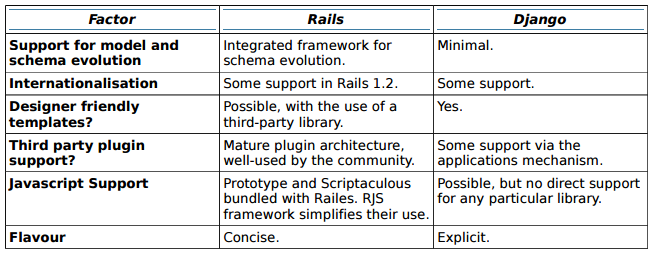
这两个框架都带有协助开发过程的脚本，并自动执行重复任务，如模式创建和执行单元测试。 Rails利用Rake，Ruby生成语言来自动执行重复任务。

Capistrano是一款部署工具，用于自动化Rails应用程序的部署。它通过ssh在远程服务器上执行命令，执行诸如检查存储库中代码的最新版本，运行迁移和加载数据等任务。它在部署到多个生产服务器时特别有用，因为它可以并行地在多个服务器上执行命令。在这个阶段，Django没有这样的工具，虽然Capistrano被其他人用于部署非Rails应用程序，但似乎没有理由不能用于部署Django应用程序。

4结论

Django和Rails旨在以类似的方式使用类似的架构来解决类似的问题。经验丰富的Rails开发团队转而使用Django或经验丰富的Django开发团队切换到Rails，并没有明确的技术优势。对于目前尚未使用Django或Rails的开发人员来说，最重要的考虑因素是实现语言。 Ruby开发人员将受益于使用Rails，而Python开发人员将受益于使用Django，允许他们应用已经拥有的技能。

对于既不了解（或两种）语言的开发人员，“最佳”框架将取决于开发环境和应用程序类型。下表总结了我们在本文中调查的方面：



在这两个框架之间进行选择可能是困难的，好消息也是这样框架是希望开发Web应用程序的团队的不错选择。

5 参考

1. [AUTOADMIN] – http://code.trebex.net/auto-admin

2. [CROFT] – http://www2.jeffcroft.com/2006/may/02/django-non-programmers/

3. [DJANGOAPI] – http://www.djangoproject.com/documentation/api\_stability/

4. [DJANGOCORE] – http://www.djangoproject.com/documentation/faq#who-s-behind-this

5. [FOWLER] – Rails Recipes, Chad Fowler

6. [FOWLER, M] – Patterns of Enterprise Application Architecture, Martin Fowler

7. [GETTEXT] – http://www.gnu.org/software/gettext/

8. [GOOGLEBATTLE] – http://www.googlebattle.com/

9. [ICEROCKET] – http://www.icerocket.com/

The Python Papers, Volume 2, Issue 2 53

10. [JOBSERVE] – http://www.jobserve.com/

11. [PYAMAZON] – http://www.josephson.org/projects/pyamazon/

12. [RAILSCORE] – http://rubyonrails.org/core

13. [RAILSMULTIBYTE] – A Rails Unicode primer,

http://fngtps.com/projects/multibyte\_for\_rails/wiki/UnicodePrimer

14. [RSPEC] – http://rspec.rubyforge.org/tools/rails.html

15. [RUBYAMAZON] – http://www.caliban.org/ruby/ruby-amazon.shtml

16. [SEEK] – http://www.seek.com.au

17. [STREAMLINED] – http://streamlines.relevancellc.com/

18. [TAGGABLE] – http://wiki.rubyonrails.org/rails/pages/ActsAsTaggablePluginHowto

19. [TECHNORATI] – http://www.technorati.com/

20. [VIKLUND] – http://andreasviklund.com/templates/

21. [WHEELER] – David A. Wheeler's Sloccount tool, http://www.dwheeler.com/sloccount

22. [YAML] – Yet Another Markup Language, http://www.yaml.org/