专业人士笔记®

Ruby on Rails

专业人士笔记

200+ 页

专业提示和技巧

Ruby® on Rails

Notes for Professionals

200+ pages

of professional hints and tricks

目录

关于 . . . 1
第1章：Ruby on Rails入门 . . . 2
第1.1节：创建一个Ruby on Rails应用程序 . . . 2
第1.2节：使用您选择的数据库创建一个新的Rails应用程序，并包含RSpec测试工具 . . . 4
第1.3节：生成控制器 . . . 4
第1.4节：安装Rails . . . 5
第1.5节：使用非标准数据库适配器创建新的Rails应用 . . . 7
第1.6节：创建JSON格式的Rails API . . . 7
第1.7节：生成带支架的资源 . . . 8
第2章：路由 . . . 10
第2.1节：资源路由（基础） . . . 10
第2.2节：约束条件 . . . 11
第2.3节：范围路由 . . . 13
第2.4节：关注点 . . . 15
第2.5节：根路由 . . . 16
第2.6节：将路由拆分到多个文件 . . . 16
第2.7节：额外的RESTful操作 . . . 17
第2.8节：成员路由和集合路由 . . . 17
第2.9节：挂载另一个应用程序 . . . 18
第2.10节：嵌套路由 . . . 18
第2.11节：重定向 . . . 19
第2.12节：重定向和通配符路由 . . . 19
第2.13节：可用区域设置范围 . . . 19
第2.14节：带有句点的URL参数 . . . 20
第3章：ActiveRecord . . . 21
第3.1节：通过生成器创建模型 . . . 21
第3.2节：回调简介 . . . 21
第3.3节：手动创建模型 . . . 22
第3.4节：手动测试你的模型 . . . 23
第3.5节：创建迁移 . . . 23
第3.6节：使用迁移创建连接表 . . . 24
第3.7节：使用模型实例更新一行 . . . 25
第4章：视图 . . . 26
第4.1节：结构 . . . 26
第4.2节：部分 . . . 26
第4.3节：AssetTagHelper . . . 27
第4.4节：替换视图中的HTML代码 . . . 28
第4.5节：HAML——在视图中使用的另一种方式 . . . 29
第5章：ActiveRecord 迁移 . . . 31
第5.1节：向表中添加多个列 . . . 31
第5.2节：向表中添加引用列 . . . 31
第5.3节：回滚迁移 . . . 32
第5.4节：添加带索引的新列 . . . 32
第5.5节：运行特定迁移 . . . 33
第5.6节：重新执行迁移 . . . 33
第5.7节：向表中添加新列 . . . 33

Contents

About . . . 1
Chapter 1: Getting started with Ruby on Rails . . . 2
Section 1.1: Creating a Ruby on Rails Application . . . 2
Section 1.2: Create a new Rails app with your choice of database and including the RSpec Testing Tool . . . 4
Section 1.3: Generating A Controller . . . 4
Section 1.4: Installing Rails . . . 5
Section 1.5: Create a new Rails app with a non-standard database adapter . . . 7
Section 1.6: Creating Rails APIs in JSON . . . 7
Section 1.7: Generate a Resource with Scaffolds . . . 8
Chapter 2: Routing . . . 10
Section 2.1: Resource Routing (Basic) . . . 10
Section 2.2: Constraints . . . 11
Section 2.3: Scoping routes . . . 13
Section 2.4: Concerns . . . 15
Section 2.5: Root route . . . 16
Section 2.6: Split routes into multiple files . . . 16
Section 2.7: Additional RESTful actions . . . 17
Section 2.8: Member and Collection Routes . . . 17
Section 2.9: Mount another application . . . 18
Section 2.10: Nested Routes . . . 18
Section 2.11: Redirection . . . 19
Section 2.12: Redirects and Wildcard Routes . . . 19
Section 2.13: Scope available locales . . . 19
Section 2.14: URL params with a period . . . 20
Chapter 3: ActiveRecord . . . 21
Section 3.1: Creating a Model via generator . . . 21
Section 3.2: Introduction to Callbacks . . . 21
Section 3.3: Creating a Model manually . . . 22
Section 3.4: Manually Testing Your Models . . . 23
Section 3.5: Creating A Migration . . . 23
Section 3.6: Create a Join Table using Migrations . . . 24
Section 3.7: Using a model instance to update a row . . . 25
Chapter 4: Views . . . 26
Section 4.1: Structure . . . 26
Section 4.2: Partials . . . 26
Section 4.3: AssetTagHelper . . . 27
Section 4.4: Replace HTML code in Views . . . 28
Section 4.5: HAML - an alternative way to use in your views . . . 29
Chapter 5: ActiveRecord Migrations . . . 31
Section 5.1: Adding multiple columns to a table . . . 31
Section 5.2: Add a reference column to a table . . . 31
Section 5.3: Rollback migrations . . . 32
Section 5.4: Add a new column with an index . . . 32
Section 5.5: Run specific migration . . . 33
Section 5.6: Redo migrations . . . 33
Section 5.7: Add a new column to a table . . . 33

第5.8节：从表中删除现有列 . . . 34
第5.9节：添加带默认值的列 . . . 34
第5.10节：在不同环境中运行迁移 . . . 35
第5.11节：创建新表 . . . 35
第5.12节：运行迁移 . . . 35
第5.13节：更改现有列的类型 . . . 35
第5.14节：创建hstore列 . . . 36
第5.15节：创建连接表 . . . 36
第5.16节：添加自引用 . . . 37
第5.17节：创建数组列 . . . 37
第5.18节：向表中添加唯一列 . . . 37
第5.19节：检查迁移状态 . . . 38
第5.20节：更改表 . . . 38
第5.21节：向现有数据添加非空约束 . . . 38
第5.22节：禁止空值 . . . 39
第6章：Rails最佳实践 . . . 40
第6.1节：胖模型，瘦控制器 . . . 40
第6.2节：领域对象（不再有胖模型） . . . 40
第6.3节：注意default_scope . . . 42
第6.4节：约定优于配置 . . . 44
第6.5节：不要重复自己（DRY） . . . 45
第6.6节：你不会需要它（YAGNI） . . . 45
第7章：命名约定 . . . 47
第7.1节：控制器 . . . 47
第7.2节：模型 . . . 47
第7.3节：文件名和自动加载 . . . 47
第7.4节：视图和布局 . . . 48
第7.5节：从控制器名称生成模型类 . . . 48
第8章：ActionCable . . . 49
第8.1节：用户认证 . . . 49
第8.2节：[基础] 服务器端 . . . 49
第8.3节：[基础] 客户端（Coffeescript） . . . 49
第9章：ActiveModel . . . 51
第9.1节：使用 ActiveModel::Validations . . . 51
第10章：Rails中的用户认证 . . . 52
第10.1节：使用 Devise 进行认证 . . . 52
第10.2节：Devise 控制器过滤器与辅助方法 . . . 52
第10.3节：Omniauth . . . 53
第10.4节：has_secure_password . . . 53
第10.5节：has_secure_token . . . 53
第11章：ActiveRecord 关联 . . . 55
第11.1节：多态关联 . . . 55
第11.2节：自引用关联 . . . 55
第11.3节：belongs_to . . . 56
第11.4节：has_one . . . 56
第11.5节：has_many . . . 56
第11.6节：has_many :through 关联 . . . 57
第11.7节：has_one :through 关联 . . . 57
第11.8节：has_and_belongs_to_many 关联 . . . 58

Section 5.8: Remove an existing column from a table . . . 34
Section 5.9: Add column with default value . . . 34
Section 5.10: Running migrations in different environments . . . 35
Section 5.11: Create a new table . . . 35
Section 5.12: Running migrations . . . 35
Section 5.13: Change an existing column's type . . . 35
Section 5.14: Create a hstore column . . . 36
Section 5.15: Create a join table . . . 36
Section 5.16: Add a self reference . . . 37
Section 5.17: Create an array column . . . 37
Section 5.18: Add an unique column to a table . . . 37
Section 5.19: Checking migration status . . . 38
Section 5.20: Changing Tables . . . 38
Section 5.21: Adding a NOT NULL constraint to existing data . . . 38
Section 5.22: Forbid null values . . . 39
Chapter 6: Rails Best Practices . . . 40
Section 6.1: Fat Model, Skinny Controller . . . 40
Section 6.2: Domain Objects (No More Fat Models) . . . 40
Section 6.3: Beware of default_scope . . . 42
Section 6.4: Convention Over Configuration . . . 44
Section 6.5: Don't Repeat Yourself (DRY) . . . 45
Section 6.6: You Ain't Gonna Need it (YAGNI) . . . 45
Chapter 7: Naming Conventions . . . 47
Section 7.1: Controllers . . . 47
Section 7.2: Models . . . 47
Section 7.3: Filenames and autoloading . . . 47
Section 7.4: Views and Layouts . . . 48
Section 7.5: Models class from Controller name . . . 48
Chapter 8: ActionCable . . . 49
Section 8.1: User Authentication . . . 49
Section 8.2: [Basic] Server Side . . . 49
Section 8.3: [Basic] Client Side (Coffeescript) . . . 49
Chapter 9: ActiveModel . . . 51
Section 9.1: Using ActiveModel::Validations . . . 51
Chapter 10: User Authentication in Rails . . . 52
Section 10.1: Authentication using Devise . . . 52
Section 10.2: Devise Controller Filters & Helpers . . . 52
Section 10.3: Omniauth . . . 53
Section 10.4: has_secure_password . . . 53
Section 10.5: has_secure_token . . . 53
Chapter 11: ActiveRecord Associations . . . 55
Section 11.1: Polymorphic association . . . 55
Section 11.2: Self-Referential Association . . . 55
Section 11.3: belongs_to . . . 56
Section 11.4: has_one . . . 56
Section 11.5: has_many . . . 56
Section 11.6: The has_many :through association . . . 57
Section 11.7: The has_one :through association . . . 57
Section 11.8: The has_and_belongs_to_many association . . . 58

第12章：ActiveRecord 验证 59
第12.1节：验证属性的长度 59
第12.2节：验证属性的格式 59
第12.3节：验证属性的存在性 59
第12.4节：自定义验证 60
第12.5节：验证属性的包含性 60
第12.6节：分组验证 61
第12.7节：验证属性的数值性 61
第12.8节：验证属性的唯一性 62
第12.9节：跳过验证 62
第12.10节：属性确认 62
第12.11节：使用 :on 选项 63
第12.12节：条件验证 63
第13章：ActiveRecord 查询接口 64
第13.1节：.where 64
第13.2节：带数组的 .where 64
第13.3节：范围 65
第13.4节：获取第一条和最后一条记录 66
第13.5节：排序 67
第13.6节：where.not 67
第13.7节：Includes 68
第13.8节：连接（Joins） 68
第13.9节：限制和偏移（Limit and Offset） 69
第13.10节：.find_by 69
第13.11节：.delete_all 69
第13.12节：ActiveRecord 不区分大小写搜索 69
第13.13节：.group 和 .count 70
第13.14节：.distinct（或 .uniq） 70
第14章：ActionMailer 71
第14.1节：基础邮件发送器 71
第14.2节：生成新的邮件发送器 72
第14.3节：ActionMailer 拦截器 72
第14.4节：添加附件 72
第14.5节：ActionMailer 回调 73
第14.6节：生成定期通讯 73
第15章：Rails 生成命令 79
第15.1节：Rails 生成控制器 79
第15.2节：Rails 生成迁移 80
第15.3节：Rails 生成脚手架 80
第15.4节：Rails 生成模型 81
第16章：配置 82
第16.1节：自定义配置 82
第17章：国际化（I18n） 84
第17.1节：带参数的国际化 84
第17.2节：翻译ActiveRecord模型属性 84
第17.3节：从HTTP请求获取语言环境 86
第17.4节：复数形式处理 87
第17.5节：通过请求设置区域 87
第17.6节：在HTML标签和符号中使用I18n 89
第17.7节：在视图中使用I18n 89

Chapter 12: ActiveRecord Validations 59
Section 12.1: Validating length of an attribute 59
Section 12.2: Validates format of an attribute 59
Section 12.3: Validating presence of an attribute 59
Section 12.4: Custom validations 60
Section 12.5: Validates inclusion of an attribute 60
Section 12.6: Grouping validation 61
Section 12.7: Validating numericality of an attribute 61
Section 12.8: Validate uniqueness of an attribute 62
Section 12.9: Skipping Validations 62
Section 12.10: Confirmation of attribute 62
Section 12.11: Using :on option 63
Section 12.12: Conditional validation 63
Chapter 13: ActiveRecord Query Interface 64
Section 13.1: .where 64
Section 13.2: .where with an array 64
Section 13.3: Scopes 65
Section 13.4: Get first and last record 66
Section 13.5: Ordering 67
Section 13.6: where.not 67
Section 13.7: Includes 68
Section 13.8: Joins 68
Section 13.9: Limit and Offset 69
Section 13.10: .find_by 69
Section 13.11: .delete_all 69
Section 13.12: ActiveRecord case insensitive search 69
Section 13.13: .group and .count 70
Section 13.14: .distinct (or .uniq) 70
Chapter 14: ActionMailer 71
Section 14.1: Basic Mailer 71
Section 14.2: Generating a new mailer 72
Section 14.3: ActionMailer Interceptor 72
Section 14.4: Adding Attachments 72
Section 14.5: ActionMailer Callbacks 73
Section 14.6: Generate a Scheduled Newsletter 73
Chapter 15: Rails generate commands 79
Section 15.1: Rails Generate Controller 79
Section 15.2: Rails Generate Migration 80
Section 15.3: Rails Generate Scaffold 80
Section 15.4: Rails Generate Model 81
Chapter 16: Configuration 82
Section 16.1: Custom configuration 82
Chapter 17: I18n - Internationalization 84
Section 17.1: I18n with arguments 84
Section 17.2: Translating ActiveRecord model attributes 84
Section 17.3: Get locale from HTTP request 86
Section 17.4: Pluralization 87
Section 17.5: Set locale through requests 87
Section 17.6: Use I18n with HTML Tags and Symbols 89
Section 17.7: Use I18n in views 89

第18章：在Rails中使用Google地图 . . . 91
第18.1节：将Google地图JavaScript标签添加到布局头部 . . . 91
第18.2节：对模型进行地理编码 . . . 91
第18.3节：在个人资料视图中显示Google地图上的地址 . . . 92
第18.4节：使用JavaScript在地图上设置标记 . . . 93
第18.5节：使用CoffeeScript类初始化地图 . . . 94
第18.6节：使用CoffeeScript类初始化地图标记 . . . 94
第18.7节：使用CoffeeScript类自动缩放地图 . . . 95
第18.8节：将模型属性以json格式暴露 . . . 96
第19章：文件上传 . . . 98
第19.1节：使用Carrierwave进行单文件上传 . . . 98
第19.2节：嵌套模型 - 多文件上传 . . . 98
第20章：缓存 . . . 100
第20.1节：俄罗斯套娃缓存 . . . 100
第20.2节：SQL缓存 . . . 100
第20.3节：动作缓存 . . . 101
第20.4节：片段缓存 . . . 101
第20.5节：页面缓存 . . . 102
第20.6节：HTTP缓存 . . . 102
第21章：ActionController（动作控制器） . . . 103
第21.1节：基础REST控制器 . . . 103
第21.2节：过滤器 . . . 104
第21.3节：生成控制器 . . . 106
第21.4节：使用redirect_to捕获ActiveRecord::RecordNotFound异常 . . . 107
第21.5节：为异常显示错误页面 . . . 107
第21.6节：输出JSON而非HTML . . . 108
第21.7节：控制器（基础） . . . 108
第21.8节：参数 . . . 109
第21.9节：过滤参数（基础） . . . 109
第21.10节：重定向 . . . 110
第21.11节：使用视图 . . . 110
第22章：配置 . . . 112
第22.1节：Rails通用配置 . . . 112
第22.2节：配置资产 . . . 112
第22.3节：配置生成器 . . . 112
第22.4节：Rails中的环境 . . . 113
第22.5节：数据库配置 . . . 113
第23章：安全常量化（Safe Constantize） . . . 115
第23.1节：成功的safe_constantize . . . 115
第23.2节：失败的safe_constantize . . . 115
第24章：Rails 5 . . . 116
第24.1节：如何在RVM上安装Ruby on Rails 5 . . . 116
第24.2节：创建Ruby on Rails 5 API . . . 116
第25章：使用CanCan进行授权 . . . 119
第25.1节：CanCan入门 . . . 119
第25.2节：处理大量权限 . . . 119
第25.3节：定义权限 . . . 121
第25.4节：快速测试权限 . . . 121
第26章：Mongoid . . . 123

Chapter 18: Using GoogleMaps with Rails . . . 91
Section 18.1: Add the google maps javascript tag to the layout header . . . 91
Section 18.2: Geocode the model . . . 91
Section 18.3: Show addresses on a google map in the profile view . . . 92
Section 18.4: Set the markers on the map with javascript . . . 93
Section 18.5: Initialize the map using a coffee script class . . . 94
Section 18.6: Initialize the map markers using a coffee script class . . . 94
Section 18.7: Auto-zoom a map using a coffee script class . . . 95
Section 18.8: Exposing the model properties as json . . . 96
Chapter 19: File Uploads . . . 98
Section 19.1: Single file upload using Carrierwave . . . 98
Section 19.2: Nested model - multiple uploads . . . 98
Chapter 20: Caching . . . 100
Section 20.1: Russian Doll Caching . . . 100
Section 20.2: SQL Caching . . . 100
Section 20.3: Action caching . . . 101
Section 20.4: Fragment caching . . . 101
Section 20.5: Page caching . . . 102
Section 20.6: HTTP caching . . . 102
Chapter 21: ActionController . . . 103
Section 21.1: Basic REST Controller . . . 103
Section 21.2: Filters . . . 104
Section 21.3: Generating a controller . . . 106
Section 21.4: Rescuing ActiveRecord::RecordNotFound with redirect_to . . . 107
Section 21.5: Display error pages for exceptions . . . 107
Section 21.6: Output JSON instead of HTML . . . 108
Section 21.7: Controllers (Basic) . . . 108
Section 21.8: Parameters . . . 109
Section 21.9: Filtering parameters (Basic) . . . 109
Section 21.10: Redirecting . . . 110
Section 21.11: Using Views . . . 110
Chapter 22: Configuration . . . 112
Section 22.1: Rails General Configuration . . . 112
Section 22.2: Configuring assets . . . 112
Section 22.3: Configuring generators . . . 112
Section 22.4: Environments in Rails . . . 113
Section 22.5: Database Configuration . . . 113
Chapter 23: Safe Constantize . . . 115
Section 23.1: Successful safe_constantize . . . 115
Section 23.2: Unsuccessful safe_constantize . . . 115
Chapter 24: Rails 5 . . . 116
Section 24.1: How to install Ruby on Rails 5 on RVM . . . 116
Section 24.2: Creating a Ruby on Rails 5 API . . . 116
Chapter 25: Authorization with CanCan . . . 119
Section 25.1: Getting started with CanCan . . . 119
Section 25.2: Handling large number of abilities . . . 119
Section 25.3: Defining abilities . . . 121
Section 25.4: Quickly test an ability . . . 121
Chapter 26: Mongoid . . . 123

第26.1节：字段 . . . 123
第26.2节：安装 . . . 123
第26.3节：创建模型 . . . 123
第26.4节：经典关联 . . . 124
第26.5节：嵌入式关联 . . . 124
第26.6节：数据库调用 . . . 125
第27章：宝石 . . . 126
第27.1节：Gemfiles（宝石文件） . . . 126
第27.2节：什么是宝石？ . . . 126
第27.3节：Bundler（打包工具） . . . 127
第27.4节：Gemsets（宝石集） . . . 127
第28章：更改默认时区 . . . 130
第28.1节：更改Rails时区并让Active Record在该时区存储时间 . . . 130
第28.2节：更改Rails时区，但继续让Active Record以UTC保存到数据库 . . . 130
第29章：资源管线 . . . 131
第29.1节：清单文件和指令 . . . 131
第29.2节：Rake任务 . . . 132
第29.3节：基本用法 . . . 132
第30章：升级Rails . . . 133
第30.1节：从Rails 4.2升级到Rails 5.0 . . . 133
第31章：ActiveRecord锁定 . . . 135
第31.1节：乐观锁定 . . . 135
第31.2节：悲观锁定 . . . 135
第32章：调试 . . . 136
第32.1节：调试Rails应用程序 . . . 136
第32.2节：快速调试Ruby on Rails + 初学者建议 . . . 136
第32.3节：使用pry调试ruby-on-rails应用程序 . . . 138
第32.4节：在你的IDE中调试 . . . 139
第33章：配置Angular与Rails . . . 141
第33.1节：Angular与Rails入门 . . . 141
第34章：Rails日志记录器 . . . 144
第34.1节：Rails.logger . . . 144
第35章：Prawn PDF . . . 145
第35.1节：高级示例 . . . 145
第35.2节：基础示例 . . . 146
第36章：Rails API . . . 147
第36.1节：创建仅API应用程序 . . . 147
第37章：在Heroku上部署Rails应用 . . . 148
第37.1节：部署您的应用程序 . . . 148
第37.2节：管理Heroku的生产和预发布环境 . . . 150
第38章：ActiveSupport . . . 152
第38.1节：核心扩展：字符串访问 . . . 152
第38.2节：核心扩展：字符串到日期/时间的转换 . . . 153
第38.3节：核心扩展：字符串排除 . . . 153
第38.4节：核心扩展：字符串过滤器 . . . 153
第38.5节：核心扩展：字符串屈折变化 . . . 155
第39章：表单助手 . . . 158

Section 26.1: Fields . . . 123
Section 26.2: Installation . . . 123
Section 26.3: Creating a Model . . . 123
Section 26.4: Classic Associations . . . 124
Section 26.5: Embedded Associations . . . 124
Section 26.6: Database Calls . . . 125
Chapter 27: Gems . . . 126
Section 27.1: Gemfiles . . . 126
Section 27.2: What is a gem? . . . 126
Section 27.3: Bundler . . . 127
Section 27.4: Gemsets . . . 127
Chapter 28: Change default timezone . . . 130
Section 28.1: Change Rails timezone AND have Active Record store times in this timezone . . . 130
Section 28.2: Change Rails timezone, but continue to have Active Record save in the database in UTC . . . 130
Chapter 29: Asset Pipeline . . . 131
Section 29.1: Manifest Files and Directives . . . 131
Section 29.2: Rake tasks . . . 132
Section 29.3: Basic Usage . . . 132
Chapter 30: Upgrading Rails . . . 133
Section 30.1: Upgrading from Rails 4.2 to Rails 5.0 . . . 133
Chapter 31: ActiveRecord Locking . . . 135
Section 31.1: Optimistic Locking . . . 135
Section 31.2: Pessimistic Locking . . . 135
Chapter 32: Debugging . . . 136
Section 32.1: Debugging Rails Application . . . 136
Section 32.2: Debugging Ruby on Rails Quickly + Beginner advice . . . 136
Section 32.3: Debugging ruby-on-rails application with pry . . . 138
Section 32.4: Debugging in your IDE . . . 139
Chapter 33: Configure Angular with Rails . . . 141
Section 33.1: Angular with Rails 101 . . . 141
Chapter 34: Rails logger . . . 144
Section 34.1: Rails.logger . . . 144
Chapter 35: Prawn PDF . . . 145
Section 35.1: Advanced Example . . . 145
Section 35.2: Basic Example . . . 146
Chapter 36: Rails API . . . 147
Section 36.1: Creating an API-only application . . . 147
Chapter 37: Deploying a Rails app on Heroku . . . 148
Section 37.1: Deploying your application . . . 148
Section 37.2: Managing Production and staging environments for a Heroku . . . 150
Chapter 38: ActiveSupport . . . 152
Section 38.1: Core Extensions: String Access . . . 152
Section 38.2: Core Extensions: String to Date/Time Conversion . . . 153
Section 38.3: Core Extensions: String Exclusion . . . 153
Section 38.4: Core Extensions: String Filters . . . 153
Section 38.5: Core Extensions: String Inflection . . . 155
Chapter 39: Form Helpers . . . 158

第39.1节：创建搜索表单 . 158
第39.2节：下拉菜单 . 158
第39.3节：表单元素助手 . 158
第40章：ActiveRecord事务 . 161
第40.1节：基本示例 . 161
第40.2节：单个事务中的不同ActiveRecord类 . 161
第40.3节：多数据库连接 . 161
第40.4节：save和destroy自动包装在事务中 . 161
第40.5节：回调 . 162
第40.6节：回滚事务 . 162
第41章：RSpec与Ruby on Rails . 163
第41.1节：安装RSpec . 163
第42章：装饰器模式 . 164
第42.1节：使用Draper装饰模型 . 164
第42.2节：使用SimpleDelegator装饰模型 . 164
第43章：Elasticsearch . 166
第43.1节：Searchkick . 166
第43.2节：安装与测试 . 166
第43.3节：开发工具的设置 . 167
第43.4节：介绍 . 167
第44章：使用 react-rails gem 在 Rails 中集成 React . 168
第44.1节：使用 rails_react gem 在 Rails 中安装 React . 168
第44.2节：在应用程序中使用 react_rails . 168
第44.3节：渲染与挂载 . 169
第45章：Rails秘籍——高级Rails配方/学习和编码技巧
. 170
第45.1节：使用Rails控制台操作表格 . 170
第45.2节：Rails方法——返回布尔值 . 170
第45.3节：处理错误——未定义方法`where`，针对#<Array:0x000000071923f8> . 171
第46章：多用途ActiveRecord列 . 172
第46.1节：保存对象 . 172
第46.2节：操作方法 . 172
第47章：类的组织 . 173
第47.1节：服务类 . 173
第47.2节：模型类 . 175
第48章：浅层路由 . 176
第48.1节：浅层的使用 . 176
第49章：模型状态：AASM . 177
第49.1节：使用AASM的基本状态 . 177
第50章：Rails 5 API 认证 . 179
第50.1节：使用Rails的authenticate_with_http_token进行认证 . 179
第51章：测试Rails应用程序 . 180
第51.1节：单元测试 . 180
第51.2节：请求测试 . 180
第52章：活动任务 . 181
第52.1节：介绍 . 181
第52.2节：示例任务 . 181
第52.3节：通过生成器创建活动任务 . 181

Section 39.1: Creating a search form . 158
Section 39.2: Dropdown . 158
Section 39.3: Helpers for form elements . 158
Chapter 40: ActiveRecord Transactions . 161
Section 40.1: Basic example . 161
Section 40.2: Different ActiveRecord classes in a single transaction . 161
Section 40.3: Multiple database connections . 161
Section 40.4: save and destroy are automatically wrapped in a transaction . 161
Section 40.5: Callbacks . 162
Section 40.6: Rolling back a transaction . 162
Chapter 41: RSpec and Ruby on Rails . 163
Section 41.1: Installing RSpec . 163
Chapter 42: Decorator pattern . 164
Section 42.1: Decorating a Model using Draper . 164
Section 42.2: Decorating a Model using SimpleDelegator . 164
Chapter 43: Elasticsearch . 166
Section 43.1: Searchkick . 166
Section 43.2: Installation and testing . 166
Section 43.3: Setting up tools for development . 167
Section 43.4: Introduction . 167
Chapter 44: React with Rails using react-rails gem . 168
Section 44.1: React installation for Rails using rails_react gem . 168
Section 44.2: Using react_rails within your application . 168
Section 44.3: Rendering & mounting . 169
Chapter 45: Rails Cookbook - Advanced rails recipes/learnings and coding techniques
. 170
Section 45.1: Playing with Tables using rails console . 170
Section 45.2: Rails methods - returning boolean values . 170
Section 45.3: Handling the error - undefined method `where' for #<Array:0x000000071923f8> . 171
Chapter 46: Multipurpose ActiveRecord columns . 172
Section 46.1: Saving an object . 172
Section 46.2: How To . 172
Chapter 47: Class Organization . 173
Section 47.1: Service Class . 173
Section 47.2: Model Class . 175
Chapter 48: Shallow Routing . 176
Section 48.1: Use of shallow . 176
Chapter 49: Model states: AASM . 177
Section 49.1: Basic state with AASM . 177
Chapter 50: Rails 5 API Authetication . 179
Section 50.1: Authentication with Rails authenticate_with_http_token . 179
Chapter 51: Testing Rails Applications . 180
Section 51.1: Unit Test . 180
Section 51.2: Request Test . 180
Chapter 52: Active Jobs . 181
Section 52.1: Introduction . 181
Section 52.2: Sample Job . 181
Section 52.3: Creating an Active Job via the generator . 181

第53章：多年来的Rails框架 . . . 182
第53.1节：如何查找当前版本Rails中可用的框架？ . . . 182
第53.2节：Rails 1.x版本 . . . 182
第53.3节：Rails 2.x中的Rails框架 . . . 182
第53.4节：Rails 3.x中的Rails框架 . . . 182
第54章：Ruby on Rails中的嵌套表单 . . . 183
第54.1节：如何在Ruby on Rails中设置嵌套表单 . . . 183
第55章：Factory Girl . . . 184
第55.1节：定义工厂 . . . 184
第56章：从指定文件夹导入整个CSV文件 . . . 185
第56.1节：通过控制台命令上传CSV . . . 185
第57章：Ruby on Rails代码优化和清理工具 . . . 186
第57.1节：如果你想保持代码的可维护性、安全性和优化性，可以看看一些用于代码优化和清理的gem： . . . 186
第58章：ActiveJob . . . 187
第58.1节：创建任务 . . . 187
第58.2节：将作业加入队列 . . . 187
第59章：主动模型序列化器 . . . 188
第59.1节：使用序列化器 . . . 188
第60章：Rails引擎 - 模块化Rails . . . 189
第60.1节：创建模块化应用 . . . 189
第61章：单表继承 . . . 192
第61.1节：基本示例 . . . 192
第61.2节：自定义继承列 . . . 192
第61.3节：带有类型列且无STI的Rails模型 . . . 193
第62章：ActiveRecord事务 . . . 194
第62.1节：Active Record事务入门 . . . 194
第63章：Turbolinks . . . 195
第63.1节：绑定到turbolink的页面加载概念 . . . 195
第63.2节：在特定链接上禁用turbolinks . . . 195
第63.3节：理解应用访问 . . . 196
第63.4节：在访问开始前取消访问 . . . 196
第63.5节：跨页面加载保持元素状态 . . . 197
第64章：友好ID . . . 198
第64.1节：Rails快速入门 . . . 198
第65章：安全存储认证密钥 . . . 200
第65.1节：使用Figaro存储认证密钥 . . . 200
第66章：使用Devise进行API认证 . . . 201
第66.1节：入门 . . . 201
第67章：使用Hyperloop将React.js与Rails集成 . . . 203
第67.1节：向你的Rails应用添加一个简单的React组件（用Ruby编写） . . . 203
第67.2节：回调 . . . 203
第67.3节：声明组件参数（props） . . . 203
第67.4节：HTML标签 . . . 204
第67.5节：事件处理程序 . . . 204
第67.6节：状态 . . . 204
第68章：更改默认的Rails应用环境 . . . 205
第68.1节：在本地机器上运行 . . . 205

Chapter 53: Rails frameworks over the years . . . 182
Section 53.1: How to find what frameworks are available in the current version of Rails? . . . 182
Section 53.2: Rails versions in Rails 1.x . . . 182
Section 53.3: Rails frameworks in Rails 2.x . . . 182
Section 53.4: Rails frameworks in Rails 3.x . . . 182
Chapter 54: Nested form in Ruby on Rails . . . 183
Section 54.1: How to setup a nested form in Ruby on Rails . . . 183
Chapter 55: Factory Girl . . . 184
Section 55.1: Defining Factories . . . 184
Chapter 56: Import whole CSV files from specific folder . . . 185
Section 56.1: Uploads CSV from console command . . . 185
Chapter 57: Tools for Ruby on Rails code optimization and cleanup . . . 186
Section 57.1: If you want to keep your code maintainable, secure and optimized, look at some gems for code optimization and cleanup : . . . 186
Chapter 58: ActiveJob . . . 187
Section 58.1: Create the Job . . . 187
Section 58.2: Enqueue the Job . . . 187
Chapter 59: Active Model Serializers . . . 188
Section 59.1: Using a serializer . . . 188
Chapter 60: Rails Engine - Modular Rails . . . 189
Section 60.1: Create a modular app . . . 189
Chapter 61: Single Table Inheritance . . . 192
Section 61.1: Basic example . . . 192
Section 61.2: Custom inheritance column . . . 192
Section 61.3: Rails model with type column and without STI . . . 193
Chapter 62: ActiveRecord Transactions . . . 194
Section 62.1: Getting Started with Active Record Transactions . . . 194
Chapter 63: Turbolinks . . . 195
Section 63.1: Binding to turbolink's concept of a page load . . . 195
Section 63.2: Disable turbolinks on specific links . . . 195
Section 63.3: Understanding Application Visits . . . 196
Section 63.4: Cancelling visits before they begin . . . 196
Section 63.5: Persisting elements across page loads . . . 197
Chapter 64: Friendly ID . . . 198
Section 64.1: Rails Quickstart . . . 198
Chapter 65: Securely storing authentication keys . . . 200
Section 65.1: Storing authentication keys with Figaro . . . 200
Chapter 66: Authenticate Api using Devise . . . 201
Section 66.1: Getting Started . . . 201
Chapter 67: Integrating React.js with Rails Using Hyperloop . . . 203
Section 67.1: Adding a simple react component (written in ruby) to your Rails app . . . 203
Section 67.2: Callbacks . . . 203
Section 67.3: Declaring component parameters (props) . . . 203
Section 67.4: HTML Tags . . . 204
Section 67.5: Event Handlers . . . 204
Section 67.6: States . . . 204
Chapter 68: Change a default Rails application enviornment . . . 205
Section 68.1: Running on a local machine . . . 205

第68.2节：在服务器上运行 205
第69章：Rails引擎 206
第69.1节：著名的例子有 206
第70章：向你的Rails应用添加Amazon RDS 207
第70.1节：考虑将MYSQL RDS与Rails应用连接 207
第71章：Rails中的支付功能 208
第71.1节：如何集成Stripe 208
第72章：Docker上的Rails 210
第72.1节：Docker和docker-compose 210
附录 A：保留字 212
A.1 节：保留字列表 212
鸣谢 218
你可能也喜欢 222

Section 68.2: Running on a server 205
Chapter 69: Rails -Engines 206
Section 69.1: Famous examples are 206
Chapter 70: Adding an Amazon RDS to your rails application 207
Section 70.1: Consider we are connecting MYSQL RDS with your rails application 207
Chapter 71: Payment feature in rails 208
Section 71.1: How to integrate with Stripe 208
Chapter 72: Rails on docker 210
Section 72.1: Docker and docker-compose 210
Appendix A: Reserved Words 212
Section A.1: Reserved Word List 212
Credits 218
You may also like 222

关于

请随意免费分享此PDF，
本书的最新版本可从以下网址下载：
https://goalkicker.com/RubyOnRailsBook

请将反馈和更正发送至web@petercv.com

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/RubyOnRailsBook

Please send feedback and corrections to web@petercv.com

第1章：Ruby on Rails入门

版本发布日期

5.1.5	2018-02-14
5.1.2	2017-06-26
5.0	2016-06-30
4.2	2014-12-19
4.1	2014-04-08
4.0	2013-06-25
3.2	2012-01-20
3.1	2011-08-31
3.0	2010-08-29
2.3	2009-03-16
2.0	2007-12-07
1.2	2007-01-19
1.1	2006-03-28
1.0	2005-12-13

第1.1节：创建一个Ruby on Rails应用程序

本示例假设Ruby和Ruby on Rails已经正确安装。如果没有，可以在第1.4节：安装Rails中找到相关方法。

打开命令行或终端。要生成一个新的rails应用程序，使用rails new命令，后面跟上你的应用程序名称：

```
$ rails new my_app
```

如果你想使用特定版本的 Rails 来创建你的 Rails 应用程序，可以在生成应用程序时指定版本。为此，使用 rails _version_ new，后面跟上应用程序名称：

```
$ rails _4.2.0_ new my_app
```

这将创建一个名为 MyApp 的 Rails 应用程序，位于 my_app 目录中，并使用 bundle install 安装 Gemfile 中已列出的 gem 依赖。

要切换到新创建的应用程序目录，使用 cd 命令，表示“切换目录”。

```
$ cd my_app
```

my_app 目录包含了构成 Rails 应用程序结构的多个自动生成的文件和文件夹。以下是默认创建的文件和文件夹列表：

文件/文件夹	用途
app/	包含应用程序的控制器、模型、视图、助手、邮件发送器和资源文件。
bin/	包含启动您的应用程序的 rails 脚本，并且可以包含您用来设置、更新、部署或运行应用程序的其他脚本。
config/	配置您的应用程序的路由、数据库等。

Chapter 1: Getting started with Ruby on Rails

Version Release Date

5.1.5	2018-02-14
5.1.2	2017-06-26
5.0	2016-06-30
4.2	2014-12-19
4.1	2014-04-08
4.0	2013-06-25
3.2	2012-01-20
3.1	2011-08-31
3.0	2010-08-29
2.3	2009-03-16
2.0	2007-12-07
1.2	2007-01-19
1.1	2006-03-28
1.0	2005-12-13

Section 1.1: Creating a Ruby on Rails Application

This example assumes *Ruby* and *Ruby on Rails* have already been installed properly. If not, you can find how to do it in Section 1.4: Installing Rails.

Open up a command line or terminal. To generate a new rails application, use rails new command followed by the name of your application:

```
$ rails new my_app
```

If you want to create your Rails application with a specific Rails version then you can specify it at the time of generating the application. To do that, use `rails _version_ new` followed by the application name:

```
$ rails _4.2.0_ new my_app
```

This will create a Rails application called `MyApp` in a `my_app` directory and install the gem dependencies that are already mentioned in `Gemfile` using `bundle install`.

To switch to your newly created app's directory, use the `cd` command, which stands for `change directory`.

```
$ cd my_app
```

The `my_app` directory has a number of auto-generated files and folders that make up the structure of a Rails application. Following is a list of files and folders that are created by default:

File/Folder	Purpose
app/	Contains the controllers, models, views, helpers, mailers and assets for your application.
bin/	Contains the rails script that starts your app and can contain other scripts you use to setup, update, deploy or run your application.
config/	Configure your application's routes, database, and more.

config.ru	用于基于Rack的服务器启动应用程序的Rack配置。
db/	包含您当前的数据库模式以及数据库迁移文件。
Gemfile Gemfile.lock	这些文件允许您指定Rails应用程序所需的gem依赖。 这些文件由Bundler gem使用。
lib/	为您的应用程序扩展的模块。
log/	应用程序日志文件。
public/	唯一被外界直接看到的文件夹。包含静态文件和已编译的资源。
Rakefile	此文件定位并加载可从命令行运行的任务。任务定义 分布在Rails的各个组件中。
README.md	这是您的应用程序的简要说明手册。您应编辑此文件，告诉他人 您的应用程序的功能、如何设置等。
test/	单元测试、夹具及其他测试设备。
temp/	临时文件（如缓存和pid文件）。
vendor/	所有第三方代码的存放位置。在典型的Rails应用中，这包括供应的gem。

现在你需要从你的database.yml文件创建一个数据库：

版本 ≥ 5.0

```
rake db:create
# 或者
rails db:create
```

版本 < 5.0

```
rake db:create
```

现在我们已经创建了数据库，需要运行迁移来设置表：

版本 ≥ 5.0

```
rake db:migrate
# 或者
rails db:migrate
```

版本 < 5.0

```
rake db:migrate
```

要启动应用程序，我们需要启动服务器：

```
$ rails server
# 或者
$ rails s
```

默认情况下，Rails 会在端口 3000 启动应用程序。要使用不同的端口号启动应用程序，我们需要像这样启动服务器，

```
$ rails s -p 3010
```

如果你在浏览器中访问 http://localhost:3000 ，你将看到一个 Rails 欢迎页面，显示你的应用程序现在正在运行。

如果出现错误，可能有几种原因：

- 配置/数据库.yml存在问题。
- 您的Gemfile中有未安装的依赖项。
- 您有未完成的迁移。运行 `rails db:migrate`
- 如果需要回退到之前的迁移，运行 `rails db:rollback`

config.ru	Rack configuration for Rack based servers used to start the application.
db/	Contains your current database schema, as well as the database migrations.
Gemfile Gemfile.lock	These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem.
lib/	Extended modules for your application.
log/	Application log files.
public/	The only folder seen by the world as-is. Contains static files and compiled assets.
Rakefile	This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails.
README.md	This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up etc
test/	Unit tests, fixtures, and other test apparatus.
temp/	Temporary files (like cache and pid files).
vendor/	A place for all third-party code. In a typical Rails application this includes vendored gems.

Now you need to create a database from your `database.yml` file:

Version ≥ 5.0

```
rake db:create
# OR
rails db:create
```

Version < 5.0

```
rake db:create
```

Now that we've created the database, we need to run migrations to set up the tables:

Version ≥ 5.0

```
rake db:migrate
# OR
rails db:migrate
```

Version < 5.0

```
rake db:migrate
```

To start the application, we need to fire up the server:

```
$ rails server
# OR
$ rails s
```

By default, rails will start the application at port 3000. To start the application with different port number, we need to fire up the server like,

```
$ rails s -p 3010
```

If you navigate to http://localhost:3000 in your browser, you will see a Rails welcome page, showing that your application is now running.

If it throws an error, there may be several possible problems:

- There is a problem with the `config/database.yml`
- You have dependencies in your `Gemfile` that have not been installed.
- You have pending migrations. Run `rails db:migrate`
- In case you move to the previous migration `rails db:rollback`

如果仍然报错，您应该检查您的 config/database.yml

第1.2节：创建一个新的Rails应用，选择您喜欢的数据库并包含RSpec测试工具

Rails默认使用 sqlite3 作为数据库，但您可以生成一个带有您选择数据库的新Rails应用。只需添加 -d 选项，后跟数据库名称。

```
$ rails new MyApp -T -d postgresql
```

这是一个（非详尽）可用数据库选项列表：

- mysql
- oracle
- postgresql
- sqlite3
- frontbase
- ibm_db
- sqlserver
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- jdbc

-T 命令表示跳过安装 minitest。要安装替代的测试套件，如RSpec，编辑 Gemfile 并添加

```
group :development, :test do
  gem 'rspec-rails',
end
```

然后从控制台运行以下命令：

```
rails generate rspec:install
```

第 1.3 节：生成控制器

要生成一个控制器（例如Posts），请从命令行或终端进入项目目录，
然后运行：

```
$ rails generate controller Posts
```

你可以通过将generate替换为g来简化这段代码，例如：

```
$ rails g controller Posts
```

如果你打开新生成的 app/controllers/posts_controller.rb 文件，你会看到一个没有任何动作的控制器：

```
class PostsController < ApplicationController
    # 空
end
```

可以通过传入控制器名称参数来创建控制器的默认方法。

If that still throws an error, then you should check your `config/database.yml`

Section 1.2: Create a new Rails app with your choice of database and including the RSpec Testing Tool

Rails uses `sqlite3` as the default database, but you can generate a new rails application with a database of your choice. Just add the `-d` option followed by the name of the database.

```
$ rails new MyApp -T -d postgresql
```

This is a (non-exhaustive) list of available database options:

- mysql
- oracle
- postgresql
- sqlite3
- frontbase
- ibm_db
- sqlserver
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- jdbc

The -T command indicate to skip the installation of minitest. To install an alternative test suite like RSpec, edit the Gemfile and add

```
group :development, :test do
  gem 'rspec-rails',
end
```

Then launch the following command from the console:

```
rails generate rspec:install
```

Section 1.3: Generating A Controller

To generate a controller (for example `Posts`), navigate to your project directory from a command line or terminal, and run:

```
$ rails generate controller Posts
```

You can shorten this code by replacing `generate` with `g`, for example:

```
$ rails g controller Posts
```

If you open up the newly generated app/controllers/**posts_controller.rb** you'll see a controller with no actions:

```
class PostsController < ApplicationController
    # empty
end
```

It's possible to create default methods for the controller by passing in controller name arguments.

```
$ rails g controller ControllerName method1 method2
```

要在模块内创建控制器，请将控制器名称指定为路径形式，如parent_module/controller_name。
例如：

```
$ rails generate controller CreditCards open debit credit close
# 或者
$ rails g controller CreditCards open debit credit close
```

这将生成以下文件：

```
控制器：app/controllers/credit_cards_controller.rb
测试：      test/controllers/credit_cards_controller_test.rb
视图：      app/views/credit_cards/debit.html.erb [...等]
助手：      app/helpers/credit_cards_helper.rb
```

控制器只是一个定义为继承自ApplicationController的类。

你将在这个类中定义方法，这些方法将成为该控制器的动作。

第1.4节：安装Rails

在Ubuntu上安装Rails

在干净的Ubuntu系统上，安装Rails应该很简单

升级Ubuntu软件包

```
sudo apt-get update
sudo apt-get upgrade
```

安装Ruby和Rails依赖

```
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-software-properties
libffi-dev
```

安装Ruby版本管理器。在这里，简单的方法是使用rbenv

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

安装 Ruby Build

```
git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build
echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
```

重启 Shell

```
exec $SHELL
```

安装 Ruby

```
rbenv install 2.3.1
```



```
$ rails g controller ControllerName method1 method2
```

To create a controller within a module, specify the controller name as a path like `parent_module/controller_name`.
For example:

```
$ rails generate controller CreditCards open debit credit close
# OR
$ rails g controller CreditCards open debit credit close
```

This will generate the following files:

```
Controller: app/controllers/credit_cards_controller.rb
Test:       test/controllers/credit_cards_controller_test.rb
Views:      app/views/credit_cards/debit.html.erb [...etc]
Helper:     app/helpers/credit_cards_helper.rb
```

A controller is simply a class that is defined to inherit from `ApplicationController`.

It's inside this class that you'll define methods that will become the actions for this controller.

Section 1.4: Installing Rails

Installing Rails on Ubuntu

On a clean ubuntu, installation of Rails should be straight forward

Upgrading ubuntu packages

```
sudo apt-get update
sudo apt-get upgrade
```

Install Ruby and Rails dependecies

```
sudo apt-get install git-core curl zlib1g-dev build-essential libssl-dev libreadline-dev libyaml-dev libsqlite3-dev sqlite3 libxml2-dev libxslt1-dev libcurl4-openssl-dev python-software-properties
libffi-dev
```

Installing ruby version manager. In this case the easy one is using rbenv

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(rbenv init -)"' >> ~/.bashrc
```

Installing Ruby Build

```
git clone https://github.com/rbenv/ruby-build.git ~/.rbenv/plugins/ruby-build
echo 'export PATH="$HOME/.rbenv/plugins/ruby-build/bin:$PATH"' >> ~/.bashrc
```

Restart Shell

```
exec $SHELL
```

Install ruby

```
rbenv install 2.3.1
```

```
rbenv global 2.3.1
rbenv rehash
```

安装轨道

```
gem install rails
```

在Windows上安装Rails

步骤 1：安装 Ruby

我们需要安装 Ruby 编程语言。我们可以使用一个名为 RubyInstaller 的预编译版本的 Ruby。

- 从 rubyinstaller.org 下载并运行 Ruby Installer。
- 运行安装程序。勾选“将 Ruby 可执行文件添加到 PATH”，然后安装。
- 要访问 Ruby，进入 Windows 菜单，点击所有程序，向下滚动到 Ruby，然后点击“使用 Ruby 启动命令提示符”。命令提示符终端将打开。如果你输入 ruby -v 并按回车，你应该能看到你安装的 Ruby 版本号。

步骤 2：Ruby 开发工具包

安装 Ruby 后，我们可以尝试安装 Rails。但 Rails 依赖的一些库需要一些构建工具才能编译，而 Windows 默认缺少这些工具。如果你在尝试安装 Rails 时看到错误 Gem::InstallError：‘[gem 名称]’ 本地 gem 需要已安装的构建工具，这就说明了这个问题。
为了解决这个问题，我们需要安装 Ruby 开发工具包。

- 下载 DevKit
- 运行安装程序。
- 我们需要指定一个文件夹来永久安装 DevKit。我建议安装在硬盘根目录下，路径为 C:\RubyDevKit。（目录名中不要使用空格。）

现在我们需要让 DevKit 工具可用于 Ruby。

- 在命令提示符中，切换到 DevKit 目录。 cd C:\RubyDevKit 或者你安装它的任何目录。
- 我们需要运行一个 Ruby 脚本来初始化 DevKit 设置。输入 ruby dk.rb init。现在我们将告诉同一个脚本将 DevKit 添加到我们的 Ruby 安装中。输入 ruby dk.rb install。

现在 DevKit 应该可以供你的 Ruby 工具在安装新库时使用了。

步骤 3: Rails

现在我们可以安装 Rails。Rails 作为一个 Ruby gem 提供。在命令提示符中输入：

```
gem install rails
```

按下回车后，gem 程序将下载并安装该版本的 Rails gem，以及 Rails 依赖的所有其他 gem。

步骤 4: Node.js

Rails 依赖的一些库需要安装 JavaScript 运行时。让我们安装 Node.js，以确保这些库能正常工作。

- 从 here 下载 Node.js 安装程序。


```
rbenv global 2.3.1
rbenv rehash
```

Installing rails

```
gem install rails
```

Installing Rails on Windows

Step 1: ***Installing Ruby***

We need Ruby programming language installed. We can use a precompiled version of Ruby called RubyInstaller.

- Download and run Ruby Installer from rubyinstaller.org.
- Run the installer. Check "Add Ruby executables to your PATH", then install.
- To access Ruby, go to the Windows menu, click All Programs, scroll down to Ruby, and click “Start Command Prompt with Ruby”. A command prompt terminal will open. If you type `ruby -v` and press Enter, you should see the Ruby version number that you installed.

Step 2: ***Ruby Development Kit***

After installing Ruby, we can try to install Rails. But some of the libraries Rails depends on need some build tools in order to be compiled, and Windows lacks those tools by default. You can identify this if you see an error while attempting to install Rails `Gem::InstallError: The ‘[gem name]’ native gem requires installed build tools.` To fix this, we need to install the Ruby Development Kit.

- Download the DevKit
- Run the installer.
- We need to specify a folder where we’re going to permanently install the DevKit. I recommend installing it in the root of your hard drive, at `C:\RubyDevKit`. (Don’t use spaces in the directory name.)

Now we need to make the DevKit tools available to Ruby.

- In your command prompt, change to the DevKit directory. `cd C:\RubyDevKit` or whatever directory you installed it in.
- We need to run a Ruby script to initialize the DevKit setup. Type `ruby dk.rb init`. Now we’ll tell that same script to add the DevKit to our Ruby installation. Type `ruby dk.rb install`.

The DevKit should now be available for your Ruby tools to use when installing new libraries.

Step 3: ***Rails***

Now we can install Rails. Rails comes as a Ruby gem. In your command prompt, type:

```
gem install rails
```

Once you press Enter, the `gem` program will download and install that version of the Rails gem, along with all the other gems Rails depends on.

Step 4: ***Node.js***

Some libraries that Rails depends on require a JavaScript runtime to be installed. Let’s install Node.js so that those libraries work properly.

- Download the Node.js installer from **here**.

- 下载完成后，打开你的下载文件夹，运行 node-v4.4.7.pkg 安装程序。
- 阅读完整的许可协议，接受条款，然后点击“下一步”完成向导的其余步骤，保持所有设置为默认。
- 可能会弹出一个窗口，询问您是否允许该应用对计算机进行更改。点击“是”。
- 安装完成后，您需要重启计算机，以便 Rails 能访问 Node.js。

计算机重启后，别忘了打开 Windows 菜单，点击“所有程序”，向下滚动到 Ruby，然后点击“使用 Ruby 启动命令提示符”。

第1.5节：使用非标准数据库适配器创建新的 Rails 应用

Rails 默认附带 ActiveRecord，这是一个基于同名模式的 ORM（对象关系映射）。

作为 ORM，它用于处理关系映射，更准确地说，是为您处理 SQL 请求，因此仅限于 SQL 数据库。

不过，您仍然可以使用其他数据库管理系统创建 Rails 应用：

1. 只需创建不带 active-record 的应用

```
$ rails app new MyApp --skip-active-record
```

2. 在Gemfile中添加您自己的数据库管理系统

```
gem 'mongoid', '~> 5.0'
```

3. 运行 bundle install 并按照所需数据库的安装步骤进行操作。

在此示例中，mongoid 是 MongoDB 的对象映射，且像许多为 Rails 构建的其他数据库 gem 一样，它也继承自 ActiveModel，方式与 ActiveRecord 相同，提供了许多功能的通用接口
例如验证、回调、翻译等。

其他数据库适配器包括但不限于：

- datamapper
- sequel-rails

第1.6节：创建 JSON 格式的 Rails API

本示例假设您已有创建 Rails 应用的经验。

要在 Rails 5 中创建仅 API 应用，请运行

```
rails new name-of-app --api
```

在Gemfile中添加 active_model_serializers

```
gem 'active_model_serializers'
```

在终端安装 bundle

- When the download completes, visit your downloads folder, and run the node-v4.4.7.pkg installer.
- Read the full license agreement, accept the terms, and click Next through the rest of the wizard, leaving everything at the default.
- A window may pop up asking if you want to allow the app to make changes to your computer. Click “Yes”.
- When the installation is complete, you’ll need to restart your computer so Rails can access Node.js.

Once your computer restarts, don’t forget to go to the Windows menu, click “All Programs”, scroll down to Ruby, and click “Start Command Prompt with Ruby”.

Section 1.5: Create a new Rails app with a non-standard database adapter

Rails is shipped by default with ActiveRecord, an ORM (Object Relational Mapping) derived from the pattern with the same name.

As an ORM, it is built to handle relational-mapping, and more precisely by handling SQL requests for you, hence the limitation to SQL databases only.

However, you can still create a Rails app with another database management system:

1. simply create your app without active-record

```
$ rails app new MyApp --skip-active-record
```

2. add your own database management system in Gemfile

```
gem 'mongoid', '~> 5.0'
```

3. bundle install and follow the installation steps from the desired database.

In this example, mongoid is an object mapping for MongoDB and - as many other database gems built for rails - it also inherits from ActiveModel the same way as ActiveRecord, which provides a common interface for many features such as validations, callbacks, translations, etc.

Other database adapters include, but are not limited to :

- datamapper
- sequel-rails

Section 1.6: Creating Rails APIs in JSON

This example assumes that you have experience in creating Rails applications.

To create an API-only app in Rails 5, run

```
rails new name-of-app --api
```

Add active_model_serializers in Gemfile

```
gem 'active_model_serializers'
```

install bundle in terminal

```
bundle install
```

设置 ActiveModelSerializer 适配器为 :json_api

```
# config/initializers/active_model_serializer.rb
ActiveModelSerializers.config.adapter = :json_api
Mime::Type.register "application/json", :json, %w( text/x-json application/jsonrequest
application/vnd.api+json )
```

为你的资源生成新的 scaffold

```
rails generate scaffold Task name:string description:text
```

这将生成以下文件：

控制器：app/controllers/tasks_controller.rb

```
测试:         test/models/task_test.rb
              test/controllers/tasks_controller_test.rb
路由:       资源 :tasks 已添加 在 routes.rb
迁移:   db/migrate/_create_tasks.rb
模型:       app/models/task.rb
序列化器: app/serializers/task_serializer.rb
控制器: app/controllers/tasks_controller.rb
```

第1.7节：使用脚手架生成资源

来自 guides.rubyonrails.org：

> 不是直接生成模型……让我们设置一个脚手架。Rails中的脚手架是一整套模型、
> 该模型的数据库迁移、用于操作它的控制器、用于查看和操作数据的视图，
> 以及上述每个部分的测试套件。

这是一个生成名为Task的资源的脚手架示例，带有字符串类型的名称和文本描述：

```
rails generate scaffold Task name:string description:text
```

这将生成以下文件：

```
控制器:  app/controllers/tasks_controller.rb
测试:          test/models/task_test.rb
               test/controllers/tasks_controller_test.rb
路由:       资源 :tasks 已添加 在 routes.rb
视图:        app/views/tasks
               app/views/tasks/index.html.erb
               app/views/tasks/edit.html.erb
               app/views/tasks/show.html.erb
               app/views/tasks/new.html.erb
               app/views/tasks/_form.html.erb
助手:       app/helpers/tasks_helper.rb
JS:            app/assets/javascripts/tasks.coffee
CSS:           app/assets/stylesheets/tasks.scss
               app/assets/stylesheets/scaffolds.scss
```

删除名为Task的资源由脚手架生成的文件的示例


```
bundle install
```

Set the ActiveModelSerializer adapter to use :json_api

```
# config/initializers/active_model_serializer.rb
ActiveModelSerializers.config.adapter = :json_api
Mime::Type.register "application/json", :json, %w( text/x-json application/jsonrequest
application/vnd.api+json )
```

Generate a new scaffold for your resource

```
rails generate scaffold Task name:string description:text
```

This will generate the following files:

Controller: app/controllers/tasks_controller.rb

```
Test:         test/models/task_test.rb
              test/controllers/tasks_controller_test.rb
Routes:       resources :tasks added in routes.rb
Migration:    db/migrate/_create_tasks.rb
Model:        app/models/task.rb
Serializer:   app/serializers/task_serializer.rb
Controller:   app/controllers/tasks_controller.rb
```

Section 1.7: Generate a Resource with Scaffolds

From guides.rubyonrails.org:

> Instead of generating a model directly . . . 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.

Here's an example of scaffolding a resource called Task with a string name and a text description:

```
rails generate scaffold Task name:string description:text
```

This will generate the following files:

```
Controller:   app/controllers/tasks_controller.rb
Test:         test/models/task_test.rb
              test/controllers/tasks_controller_test.rb
Routes:       resources :tasks added in routes.rb
Views:        app/views/tasks
              app/views/tasks/index.html.erb
              app/views/tasks/edit.html.erb
              app/views/tasks/show.html.erb
              app/views/tasks/new.html.erb
              app/views/tasks/_form.html.erb
Helper:       app/helpers/tasks_helper.rb
JS:           app/assets/javascripts/tasks.coffee
CSS:          app/assets/stylesheets/tasks.scss
              app/assets/stylesheets/scaffolds.scss
```

example to delete files generated by scaffold for the resource called Task

```
rails destroy scaffold Task
```


rails destroy scaffold Task

第2章：路由

Rails 路由器识别 URL 并将其分发到控制器的动作。它还可以生成路径和 URL，避免在视图中硬编码字符串。

第2.1节：资源路由（基础）

路由定义在`config/routes.rb`中。它们通常作为一组相关路由定义，使用`resources`或`resource`方法。

`resources :users` 创建以下七个路由，全部映射到UsersController的动作：

```
get       '/users',          to: 'users#index'
post      '/users',          to: 'users#create'
get       '/users/new',      to: 'users#new'
get       '/users/:id/edit', to: 'users#edit'
get       '/users/:id',      to: 'users#show'
patch/put '/users/:id',      to: 'users#update'
delete    '/users/:id',      to: 'users#destroy'
```

动作名称显示在上面to参数中的#后面。必须在app/controllers/users_controller.rb中定义同名的方法，如下所示：

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # 继续所有其他方法...
end
```

您可以使用 only 或 except 来限制生成的动作：

```
resources :users, only:   [:show]
resources :users, except: [:show, :index]
```

您可以通过运行以下命令随时查看应用程序的所有路由：

版本 < 5.0

```
$ rake routes
```

版本 ≥ 5.0

```
$ rake routes
# 或者
$ rails routes
```

```
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
user      GET    /users/:id(.:format)      users#show
          PATCH  /users/:id(.:format)      users#update
          PUT    /users/:id(.:format)      users#update
          DELETE /users/:id(.:format)      users#destroy
```


Chapter 2: Routing

The Rails router recognizes URLs and dispatches them to a controller's action. It can also generate paths and URLs, avoiding the need to hardcode strings in your views.

Section 2.1: Resource Routing (Basic)

Routes are defined in `config/routes.rb`. They are often defined as a group of related routes, using the `resources` or `resource` methods.

`resources :users` creates the following seven routes, all mapping to actions of `UsersController`:

```
get       '/users',          to: 'users#index'
post      '/users',          to: 'users#create'
get       '/users/new',      to: 'users#new'
get       '/users/:id/edit', to: 'users#edit'
get       '/users/:id',      to: 'users#show'
patch/put '/users/:id',      to: 'users#update'
delete    '/users/:id',      to: 'users#destroy'
```

Action names are shown after the # in the `to` parameter above. Methods with those same names must be defined in `app/controllers/users_controller.rb` as follows:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

  # continue with all the other methods...
end
```

You can limit the actions that gets generated with `only` or `except`:

```
resources :users, only:   [:show]
resources :users, except: [:show, :index]
```

You can view all the routes of your application at any given time by running:

Version < 5.0

```
$ rake routes
```

Version ≥ 5.0

```
$ rake routes
# OR
$ rails routes
```

```
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
user      GET    /users/:id(.:format)      users#show
          PATCH  /users/:id(.:format)      users#update
          PUT    /users/:id(.:format)      users#update
          DELETE /users/:id(.:format)      users#destroy
```

要仅查看映射到特定控制器的路由：

版本 < 5.0

```
$ rake routes -c static_pages
static_pages_home     GET    /static_pages/home(.:format)    static_pages#home
static_pages_help     GET    /static_pages/help(.:format)    static_pages#help
```

版本 ≥ 5.0

```
$ rake routes -c static_pages
static_pages_home     GET     /static_pages/home(.:format)      static_pages#home
static_pages_help     GET     /static_pages/help(.:format)      static_pages#help

# 或者

$ rails routes -c static_pages
static_pages_home     GET    /static_pages/home(.:format)    static_pages#home
static_pages_help     GET    /static_pages/help(.:format)    static_pages#help
```

你可以使用-g选项搜索路由。它会显示任何部分匹配辅助方法名、URL路径或HTTP动词的路由：

版本 < 5.0

```
$ rake routes -g new_user      # 匹配辅助方法
$ rake routes -g POST          # 匹配HTTP动词POST
```

版本 ≥ 5.0

```
$ rake routes -g new_user      # 匹配辅助方法
$ rake routes -g POST          # 匹配HTTP动词POST
# 或者
$ rails routes -g new_user     # 匹配辅助方法
$ rails routes -g POST         # 匹配HTTP动词POST
```

此外，在开发模式下运行rails server时，你可以访问一个网页，显示所有路由并带有搜索过滤器，按优先级从上到下匹配，地址为<hostname>/rails/info/routes。页面看起来像这样：

辅助方法 路径 / 网址	HTTP动词	路径 [路径匹配]	控制器#动作
users_path	GET	/users(.:format)	users#index
	POST	/users(.:format)	users#create
new_user_path	GET	/users/new(.:format)	users#new
edit_user_path	GET	/users/:id/edit(.:format)	users#edit
用户路径	GET	/users/:id(.:format)	users#show
	PATCH	/users/:id(.:format)	users#update
	PUT	/users/:id(.:format)	users#update
	DELETE	/users/:id(.:format)	users#destroy

路由可以通过在routes.rb中使用resource方法而非resources方法，仅声明对成员（而非集合）可用。使用resource时，默认不会创建index路由，只有在明确请求时才会创建，例如：

```
resource :orders, only: [:index, :create, :show]
```

第2.2节：约束条件

你可以使用约束条件来过滤可用的路由。

To see only the routes that map to a particular controller:

Version < 5.0

```
$ rake routes -c static_pages
static_pages_home     GET    /static_pages/home(.:format)    static_pages#home
static_pages_help     GET    /static_pages/help(.:format)    static_pages#help
```

Version ≥ 5.0

```
$ rake routes -c static_pages
static_pages_home     GET    /static_pages/home(.:format)    static_pages#home
static_pages_help     GET    /static_pages/help(.:format)    static_pages#help

# OR

$ rails routes -c static_pages
static_pages_home     GET    /static_pages/home(.:format)    static_pages#home
static_pages_help     GET    /static_pages/help(.:format)    static_pages#help
```

You can search through routes using the `-g` option. This shows any route that partially matches the helper method name, the URL path or the HTTP verb:

Version < 5.0

```
$ rake routes -g new_user      # Matches helper method
$ rake routes -g POST          # Matches HTTP Verb POST
```

Version ≥ 5.0

```
$ rake routes -g new_user      # Matches helper method
$ rake routes -g POST          # Matches HTTP Verb POST
# OR
$ rails routes -g new_user     # Matches helper method
$ rails routes -g POST         # Matches HTTP Verb POST
```

Additionally, when running `rails` server in development mode, you can access a web page that shows all your routes with a search filter, matched in priority from top to bottom, at **`<hostname>`**`/rails/info/routes`. It will look like this:

Helper Path / Url	HTTP Verb	Path [Path Match]	Controller#Action
users_path	GET	/users(.:format)	users#index
	POST	/users(.:format)	users#create
new_user_path	GET	/users/new(.:format)	users#new
edit_user_path	GET	/users/:id/edit(.:format)	users#edit
user_path	GET	/users/:id(.:format)	users#show
	PATCH	/users/:id(.:format)	users#update
	PUT	/users/:id(.:format)	users#update
	DELETE	/users/:id(.:format)	users#destroy

Routes can be declared available for only members (not collections) using the method `resource` instead of `resources` in `routes.rb`. With `resource`, an `index` route is not created by default, but only when explicitly asking for one like this:

```
resource :orders, only: [:index, :create, :show]
```

Section 2.2: Constraints

You can filter what routes are available using constraints.

使用约束的方法有几种，包括：

- 段约束，
- 基于请求的约束
- 高级约束

例如，一个基于请求的约束，只允许特定的IP地址访问某条路由：

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
end
```

参见其他类似示例 ActionDispatch::Routing::Mapper::Scoping。

如果你想做更复杂的事情，可以使用更高级的约束并创建一个类来封装
逻辑：

```
# lib/api_version_constraint.rb
class ApiVersionConstraint
  def initialize(version:, default:)
    @version = version
    @default = default
  end

  def version_header
    "application/vnd.my-app.v#{@version}"
  end

  def matches?(request)
    @default || request.headers["Accept"].include?(version_header)
  end
end

# config/routes.rb
require "api_version_constraint"

Rails.application.routes.draw do
namespace :v1, constraints: ApiVersionConstraint.new(version: 1, default: true) do
    resources :users # 将路由到 app/controllers/v1/users_controller.rb
  end

namespace :v2, constraints: ApiVersionConstraint.new(version: 2) do
    resources :users # 将路由到 app/controllers/v2/users_controller.rb
  end
end
```

一个表单，多个提交按钮

您还可以使用表单提交标签的值作为约束，以路由到不同的操作。如果您的表单有多个提交按钮（例如“预览”和“提交”），您可以直接在您的
`routes.rb`，而不是编写javascript来更改表单目标URL。例如，使用
commit_param_routing gem，你可以利用rails `submit_tag`

Rails submit_tag 的第一个参数允许你更改表单提交参数的值

```
# app/views/orders/mass_order.html.erb
<%= form_for(@orders, url: mass_create_order_path do |f| %>
    <!-- 这里是大表单 -->
```


There are several ways to use constraints including:

- segment constraints,
- request based constraints
- advanced constraints

For example, a requested based constraint to only allow a specific IP address to access a route:

```
constraints(ip: /127\.0\.0\.1$/) do
  get 'route', to: "controller#action"
end
```

See other similar examples ActionDispatch::Routing::Mapper::Scoping.

If you want to do something more complex you can use more advanced constraints and create a class to wrap the logic:

```
# lib/api_version_constraint.rb
class ApiVersionConstraint
  def initialize(version:, default:)
    @version = version
    @default = default
  end

  def version_header
    "application/vnd.my-app.v#{@version}"
  end

  def matches?(request)
    @default || request.headers["Accept"].include?(version_header)
  end
end

# config/routes.rb
require "api_version_constraint"

Rails.application.routes.draw do
  namespace :v1, constraints: ApiVersionConstraint.new(version: 1, default: true) do
    resources :users # Will route to app/controllers/v1/users_controller.rb
  end

  namespace :v2, constraints: ApiVersionConstraint.new(version: 2) do
    resources :users # Will route to app/controllers/v2/users_controller.rb
  end
end
```

One form, several submit buttons

You can also use the value of the submit tags of a form as a constraint to route to a different action. If you have a form with multiple submit buttons (eg "preview" and "submit"), you could capture this constraint directly in your `routes.rb`, instead of writing javascript to change the form destination URL. For example with the commit_param_routing gem you can take advantage of rails `submit_tag`

Rails `submit_tag` first parameter lets you change the value of your form commit parameter

```
# app/views/orders/mass_order.html.erb
<%= form_for(@orders, url: mass_create_order_path do |f| %>
    <!-- Big form here -->
```

```
  <%= submit_tag "预览" %>
  <%= submit_tag "提交" %>
  # => <input name="commit" type="submit" value="Preview" />
  # => <input name="commit" type="submit" value="Submit" />
  ...
<% end %>

# config/routes.rb
resources :orders do
  # 以下两个路由描述相同的POST URL，但路由到不同的动作
  post 'mass_order', on: :collection, as: 'mass_order',
约束条件：CommitParamRouting.new('提交')，动作：'批量创建' # 当用户按下
"提交"
post '批量下单'，on: :集合，
约束条件：CommitParamRouting.new('预览')，动作：'批量创建预览' # 当用户
按下"预览"
  # 注意 `as:` 只定义了一次，因为表单的路径助手是 mass_create_order_path
url
  # CommitParamRouting 只是一个类似 ApiVersionContraint 的类
end
```

第2.3节：路由作用域

Rails 提供了多种组织路由的方式。

按 URL 作用域：

```
scope 'admin' do
get '仪表盘', to: 'administration#dashboard'
  resources '员工'
end
```

这将生成以下路线

```
get       '/admin/dashboard',          to: 'administration#dashboard'
post      '/admin/employees',          to: 'employees#create'
get       '/admin/employees/new',      to: 'employees#new'
get       '/admin/employees/:id/edit', to: 'employees#edit'
get       '/admin/employees/:id',      to: 'employees#show'
patch/put '/admin/employees/:id',      to: 'employees#update'
delete    '/admin/employees/:id',      to: 'employees#destroy'
```

在服务器端，将某些视图保存在不同的子文件夹中以区分管理员视图和用户视图可能更有意义。

按模块划分的范围

```
范围 模块: :管理员 执行
  get 'dashboard', to: 'administration#dashboard'
end
```

module 在给定名称的子文件夹下查找控制器文件

```
get       '/dashboard',          to: 'admin/administration#dashboard'
```

您可以通过添加一个as参数来重命名路径助手的前缀

```
范围 'admin'，作为：:administration 执行
```



```
  <%= submit_tag "Preview" %>
  <%= submit_tag "Submit" %>
  # => <input name="commit" type="submit" value="Preview" />
  # => <input name="commit" type="submit" value="Submit" />
  ...
<% end %>

# config/routes.rb
resources :orders do
  # Both routes below describe the same POST URL, but route to different actions
  post 'mass_order', on: :collection, as: 'mass_order',
    constraints: CommitParamRouting.new('Submit'), action: 'mass_create' # when the user presses
"submit"
  post 'mass_order', on: :collection,
    constraints: CommitParamRouting.new('Preview'), action: 'mass_create_preview' # when the user
presses "preview"
  # Note the `as:` is defined only once, since the path helper is mass_create_order_path for the form
url
  # CommitParamRouting is just a class like ApiVersionContraint
end
```

Section 2.3: Scoping routes

Rails provides several ways to organize your routes.

Scope by URL:

```
scope 'admin' do
  get 'dashboard', to: 'administration#dashboard'
  resources 'employees'
end
```

This generates the following routes

```
get       '/admin/dashboard',          to: 'administration#dashboard'
post      '/admin/employees',          to: 'employees#create'
get       '/admin/employees/new',      to: 'employees#new'
get       '/admin/employees/:id/edit', to: 'employees#edit'
get       '/admin/employees/:id',      to: 'employees#show'
patch/put '/admin/employees/:id',      to: 'employees#update'
delete    '/admin/employees/:id',      to: 'employees#destroy'
```

It may make more sense, on the server side, to keep some views in a different subfolder, to separate admin views from user views.

Scope by module

```
scope module: :admin do
  get 'dashboard', to: 'administration#dashboard'
end
```

`module` looks for the controller files under the subfolder of the given name

```
get       '/dashboard',          to: 'admin/administration#dashboard'
```

You can rename the path helpers prefix by adding an `as` parameter

```
scope 'admin', as: :administration do
```

```
获取 'dashboard'
end

# => administration_dashboard_path
```

Rails 提供了一种方便的方法来完成上述所有操作，使用 namespace 方法。以下声明是等效的

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

按控制器范围

```
scope controller: :management do
  获取 'dashboard'
获取 'performance'
end
```

这将生成以下路由

```
get       '/dashboard',          to: 'management#dashboard'
get       '/performance',        to: 'management#performance'
```

浅层嵌套

资源路由接受一个:shallow选项，该选项有助于尽可能缩短URL。资源不应嵌套超过一层。避免这种情况的一种方法是创建浅层路由。目标是在不需要时省略父集合的URL段。最终结果是，只有:index、:create和:new操作生成嵌套路由，其余操作保持在各自的浅层URL上下文中。自定义浅层路由的范围有两个选项：

- :shallow_path：为成员路径添加指定的前缀参数

```
scope shallow_path: "sekret" do
  resources :articles do
resources :comments, shallow: true
  end
end
```

- :shallow_prefix：为命名辅助方法添加指定的参数

```
scope shallow_prefix: "sekret" do
  resources :articles do
resources :comments, shallow: true
  end
end
```

我们还可以通过以下方式更直观地展示shallow路由：

```
resources :auctions, shallow: true do
  resources :bids do
resources :comments
  end
```



```
  get 'dashboard'
end

# => administration_dashboard_path
```

Rails provides a convenient way to do all the above, using the `namespace` method. The following declarations are equivalent

```
namespace :admin do
end

scope 'admin', module: :admin, as: :admin
```

Scope by controller

```
scope controller: :management do
  get 'dashboard'
  get 'performance'
end
```

This generate these routes

```
get       '/dashboard',          to: 'management#dashboard'
get       '/performance',        to: 'management#performance'
```

Shallow Nesting

Resource routes accept a `:shallow` option that helps to shorten URLs where possible. Resources shouldn't be nested more than one level deep. One way to avoid this is by creating shallow routes. The goal is to leave off parent collection URL segments where they are not needed. The end result is that the only nested routes generated are for the `:index` , `:create` , and `:new` actions. The rest are kept in their own shallow URL context. There are two options for scope to custom shallow routes:

- **:shallow_path**: Prefixes member paths with a specified parameter

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

- **:shallow_prefix**: Add specified parameters to named helpers

```
scope shallow_prefix: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

We can also illustrate `shallow` routes more by:

```
resources :auctions, shallow: true do
  resources :bids do
   resources :comments
  end
```

```
end
```

或者按以下方式编码（如果你'喜欢大量使用块的话）：

```
资源 :拍卖 执行
  浅层 执行
资源 :出价 执行
      资源 :评论
    结束
  结束
结束
```

生成的路由如下：

前缀	动词	URI 模式
bid_comments	GET	/bids/:bid_id/comments(.:format)
	POST	/bids/:bid_id/comments(.:format)
new_bid_comment	GET	/bids/:bid_id/comments/new(.:format)
edit_comment	GET	/comments/:id/edit(.:format)
comment	GET	/comments/:id(.:format)
	PATCH	/comments/:id(.:format)
	PUT	/comments/:id(.:format)
	DELETE	/comments/:id(.:format)
auction_bids	GET	/auctions/:auction_id/bids(.:format)
	POST	/auctions/:auction_id/bids(.:format)
new_auction_bid	GET	/auctions/:auction_id/bids/new(.:format)
edit_bid	GET	/bids/:id/edit(.:format)
竞标	GET	/bids/:id(.:format)
	PATCH	/bids/:id(.:format)
	PUT	/bids/:id(.:format)
	DELETE	/bids/:id(.:format)
拍卖	GET	/auctions(.:format)
	POST	/auctions(.:format)
new_auction	GET	/auctions/new(.:format)
edit_auction	GET	/auctions/:id/edit(.:format)
拍卖	GET	/auctions/:id(.:format)
	PATCH	/auctions/:id(.:format)
	PUT	/auctions/:id(.:format)
	DELETE	/auctions/:id(.:format)

如果你仔细分析生成的路由，你会注意到只有在需要确定显示哪些数据时，URL 的嵌套部分才会被包含。

第2.4节：关注点

为了避免嵌套路由中的重复，关注点提供了一种共享可重用公共资源的好方法。要创建关注点，请在 routes.rb 文件中使用 concern 方法。该方法需要一个符号和一个代码块：

```
concern :commentable do
  resources :comments
```



```
end
```

alternatively coded as follows (if you're block-happy):

```
resources :auctions do
 shallow do
   resources :bids do
     resources :comments
   end
 end
end
```

The resulting routes are:

Prefix	Verb	URI Pattern
bid_comments	GET	/bids/:bid_id/comments(.:format)
	POST	/bids/:bid_id/comments(.:format)
new_bid_comment	GET	/bids/:bid_id/comments/new(.:format)
edit_comment	GET	/comments/:id/edit(.:format)
comment	GET	/comments/:id(.:format)
	PATCH	/comments/:id(.:format)
	PUT	/comments/:id(.:format)
	DELETE	/comments/:id(.:format)
auction_bids	GET	/auctions/:auction_id/bids(.:format)
	POST	/auctions/:auction_id/bids(.:format)
new_auction_bid	GET	/auctions/:auction_id/bids/new(.:format)
edit_bid	GET	/bids/:id/edit(.:format)
bid	GET	/bids/:id(.:format)
	PATCH	/bids/:id(.:format)
	PUT	/bids/:id(.:format)
	DELETE	/bids/:id(.:format)
auctions	GET	/auctions(.:format)
	POST	/auctions(.:format)
new_auction	GET	/auctions/new(.:format)
edit_auction	GET	/auctions/:id/edit(.:format)
auction	GET	/auctions/:id(.:format)
	PATCH	/auctions/:id(.:format)
	PUT	/auctions/:id(.:format)
	DELETE	/auctions/:id(.:format)

If you analyze the routes generated carefully, you'll notice that the nested parts of the URL are only included when they are needed to determine what data to display.

Section 2.4: Concerns

To avoid repetition in nested routes, concerns provide a great way of sharing common resources that are reusable. To create a concern use the method `concern` within the `routes.rb` file. The method expects a symbol and block:

```
concern :commentable do
  resources :comments
```

```
end
```

虽然这段代码本身不创建任何路由，但它允许在资源上使用 :concerns 属性。最简单的示例是：

```
resource :page, concerns: :commentable
```

等效的嵌套资源看起来像这样：

```
resource :page do
  resource :comments
end
```

这将构建例如以下路由：

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

为了使关注点有意义，必须有多个资源使用该关注点。其他资源
可以使用以下任意语法来调用该关注点：

```
resource :post, concerns: %i(commentable)
resource :blog do
concerns :commentable
end
```

第2.5节：根路由

你可以使用root方法为你的应用添加主页路由。

```
# config/routes.rb
Rails.application.routes.draw do
  root "application#index"
  # 等同于：
  # get "/", "application#index"
end

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  def index
render "homepage"
  结束
结束
```

在终端中，rake routes（Rails 5 中为 rails routes）将生成：

```
root     GET    /          application#index
```

因为主页通常是最重要的路由，且路由的优先级按出现顺序排列，
root 路由通常应放在路由文件的第一位。

第2.6节：将路由拆分到多个文件

如果你的路由文件过于庞大，可以将路由放在多个文件中，并使用
Ruby的 require_relative 方法包含每个文件：


```
end
```

While not creating any routes itself, this code allows using the `:concerns` attribute on a resource. The simplest example would be:

```
resource :page, concerns: :commentable
```

The equivalent nested resource would look like this:

```
resource :page do
  resource :comments
end
```

This would build, for example, the following routes:

```
/pages/#{page_id}/comments
/pages/#{page_id}/comments/#{comment_id}
```

For concerns to be meaningful, there must be multiple resources that utilize the concern. Additional resources could use any of the following syntax to call the concern:

```
resource :post, concerns: %i(commentable)
resource :blog do
  concerns :commentable
end
```

Section 2.5: Root route

You can add a home page route to your app with the `root` method.

```
# config/routes.rb
Rails.application.routes.draw do
  root "application#index"
  # equivalent to:
  # get "/", "application#index"
end

# app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  def index
    render "homepage"
  end
end
```

And in terminal, `rake routes` (`rails routes` in Rails 5) will produce:

```
root     GET    /          application#index
```

Because the homepage is usually the most important route, and routes are prioritized in the order they appear, the `root` route should usually be the first in your routes file.

Section 2.6: Split routes into multiple files

If your routes file is overwhelmingly big, you can put your routes in multiple files and include each of the files with Ruby's `require_relative` method:

config/routes.rb：

```
YourAppName::Application.routes.draw do
  require_relative 'routes/admin_routes'
  require_relative 'routes/sidekiq_routes'
  require_relative 'routes/api_routes'
  require_relative 'routes/your_app_routes'
end
```

config/routes/api_routes.rb：

```
YourAppName::Application.routes.draw do
  namespace :api do
    # ...
  结束
结束
```

第2.7节：额外的RESTful操作

resources :photos do member do get 'preview' end collection do get 'dashboard' end end

这将创建以下路由，除了默认的7个RESTful路由之外：

```
get       '/photos/:id/preview',          to: 'photos#preview'
get       '/photos/dashboards',           to: 'photos#dashboard'
```

如果你想对单行进行此操作，可以使用：

resources :photos do get 'preview', on: :member get 'dashboard', on: :collection end

你也可以向/new路径添加一个操作：

resources :photos do get 'preview', on: :new end

这将创建：

```
get       '/photos/new/preview',          to: 'photos#preview'
```

在向你的RESTful路由添加操作时要小心，可能你遗漏了另一个资源！

第2.8节：成员和集合路由

在资源内部定义成员块会创建一个路由，该路由可以作用于该资源的单个成员基础路由：

```
resources :posts do
  member do
get 'preview'
  结束
结束
```

这将生成以下成员路由：

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

集合路由允许创建可以作用于资源对象集合的路由：

config/routes.rb:

```
YourAppName::Application.routes.draw do
  require_relative 'routes/admin_routes'
  require_relative 'routes/sidekiq_routes'
  require_relative 'routes/api_routes'
  require_relative 'routes/your_app_routes'
end
```

config/routes/api_routes.rb:

```
YourAppName::Application.routes.draw do
  namespace :api do
    # ...
  end
end
```

Section 2.7: Additional RESTful actions

resources :photos do member do get 'preview' end collection do get 'dashboard' end end

This creates the following routes **in addition to default 7 RESTful routes**:

```
get       '/photos/:id/preview',          to: 'photos#preview'
get       '/photos/dashboards',           to: 'photos#dashboard'
```

If you want to do this for single lines, you can use:

resources :photos do get 'preview', on: :member get 'dashboard', on: :collection end

You can also add an action to the `/new` path:

resources :photos do get 'preview', on: :new end

Which will create:

```
get       '/photos/new/preview',          to: 'photos#preview'
```

Be mindful when adding actions to your RESTful routes, probably you are missing another resource!

Section 2.8: Member and Collection Routes

Defining a member block inside a resource creates a route that can act on an individual member of that resource-based route:

```
resources :posts do
  member do
    get 'preview'
  end
end
```

This generates the following member route:

```
get '/posts/:id/preview', to: 'posts#preview'
# preview_post_path
```

Collection routes allow for creating routes that can act on a collection of resource objects:

```ruby
resources :posts do
  collection do
    get 'search'
  end
end
```

这将生成以下集合路由：

```ruby
get '/posts/search', to: 'posts#search'
# search_posts_path
```

另一种语法：

```ruby
resources :posts do
  get 'preview', on: :member
  get 'search',  on: :collection
end
```

第2.9节：挂载另一个应用程序

mount 用于挂载另一个应用程序（基本上是rack应用程序）或rails引擎，以便在当前应用程序中使用

语法：

```ruby
mount SomeRackApp, at: "some_route"
```

现在你可以使用路由助手 some_rack_app_path 或 some_rack_app_url 访问上述挂载的应用程序。

但如果你想重命名这个助手名称，可以这样做：

```ruby
mount SomeRackApp, at: "some_route", as: :myapp
```

这将生成myapp_path和myapp_url辅助方法，可用于导航到此挂载的应用程序。

第2.10节：嵌套路由

如果你想添加嵌套路由，可以在 routes. b 文件中编写以下代码。

```ruby
resources :admins do
  resources :employees
end
```

这将生成以下路由：

```
admin_employees GET     /admins/:admin_id/employees(.:format)            employees#index
                POST    /admins/:admin_id/employees(.:format)            employees#create
  new_admin_employee GET    /admins/:admin_id/employees/new(.:format)        employees#new
 edit_admin_employee GET    /admins/:admin_id/employees/:id/edit(.:format)   employees#edit
      admin_employee GET    /admins/:admin_id/employees/:id(.:format)        employees#show
                PATCH   /admins/:admin_id/employees/:id(.:format)        employees#update
                PUT     /admins/:admin_id/employees/:id(.:format)        employees#update
                DELETE  /admins/:admin_id/employees/:id(.:format)        employees#destroy
             admins GET     /admins(.:format)                                admins#index
                POST    /admins(.:format)                                admins#create
          new_admin GET     /admins/new(.:format)                            admins#new
```



```ruby
resources :posts do
  collection do
    get 'search'
  end
end
```

This generates the following collection route:

```ruby
get '/posts/search', to: 'posts#search'
# search_posts_path
```

An alternate syntax:

```ruby
resources :posts do
  get 'preview', on: :member
  get 'search',  on: :collection
end
```

Section 2.9: Mount another application

mount is used to mount another application (basically rack application) or rails engines to be used within the current application

syntax:

```ruby
mount SomeRackApp, at: "some_route"
```

Now you can access above mounted application using route helper `some_rack_app_path` or `some_rack_app_url`.

But if you want to rename this helper name you can do it as:

```ruby
mount SomeRackApp, at: "some_route", as: :myapp
```

This will generate the `myapp_path` and `myapp_url` helpers which can be used to navigate to this mounted app.

Section 2.10: Nested Routes

If you want to add nested routes you can write the following code in `routes.rb` file.

```ruby
resources :admins do
  resources :employees
end
```

This will generate following routes:

```
    admin_employees GET     /admins/:admin_id/employees(.:format)            employees#index
                    POST    /admins/:admin_id/employees(.:format)            employees#create
  new_admin_employee GET    /admins/:admin_id/employees/new(.:format)        employees#new
 edit_admin_employee GET    /admins/:admin_id/employees/:id/edit(.:format)   employees#edit
      admin_employee GET    /admins/:admin_id/employees/:id(.:format)        employees#show
                    PATCH   /admins/:admin_id/employees/:id(.:format)        employees#update
                    PUT     /admins/:admin_id/employees/:id(.:format)        employees#update
                    DELETE  /admins/:admin_id/employees/:id(.:format)        employees#destroy
               admins GET     /admins(.:format)                                admins#index
                    POST    /admins(.:format)                                admins#create
          new_admin GET     /admins/new(.:format)                            admins#new
```

```
edit_admin GET     /admins/:id/edit(.:format)          admins#edit
     admin GET     /admins/:id(.:format)               admins#show
           PATCH   /admins/:id(.:format)               admins#update
           PUT     /admins/:id(.:format)               admins#update
           DELETE  /admins/:id(.:format)               admins#destroy
```

第2.11节：重定向

你可以在Rails路由中按如下方式执行重定向：

版本 ≥ 4.0

```
get '/stories', 到: redirect('/posts')
```

版本 < 4.0

```
match "/abc" => redirect("http://example.com/abc")
```

你也可以将所有未知路由重定向到指定路径：

版本 ≥ 4.0

```
match '*path' => redirect('/'), 通过: :get
# 或者
get '*path' => redirect('/')
```

版本 < 4.0

```
match '*path' => redirect('/')
```

第2.12节：重定向和通配符路由

如果你想为用户提供一个方便的URL，但直接映射到你已经使用的另一个URL，使用重定向：

```
# config/routes.rb
TestApp::Application.routes.draw do
get 'courses/:course_name' => redirect('/courses/%{course_name}/lessons'), :as => "course"
end
```

嗯，这变得很快有趣起来了。这里的基本原则是使用#redirect方法将一个路由发送到另一个路由。如果你的路由相当简单，这是一种非常直接的方法。但如果你还想传递原始参数，你需要通过捕获参数到%{here}中来做一些技巧。注意所有内容周围的单引号。

在上面的例子中，我们还通过使用带有 :as 参数的别名重命名了路由以方便使用。这让我们可以在像 #_path 这样的辅助方法中使用该名称。同样，使用$ rake routes来测试你的路由。

第2.13节：作用域可用语言环境

如果你的应用支持多种语言，通常会在URL中显示当前的语言环境。

```
scope '/(:locale)', locale: /#{I18n.available_locales.join('|')}/ do
    root 'example#root'
    # 其他路由
end
```

你的根路由将通过I18n.available_locales中定义的语言环境访问。


```
edit_admin GET     /admins/:id/edit(.:format)          admins#edit
     admin GET     /admins/:id(.:format)               admins#show
           PATCH   /admins/:id(.:format)               admins#update
           PUT     /admins/:id(.:format)               admins#update
           DELETE  /admins/:id(.:format)               admins#destroy
```

Section 2.11: Redirection

You can perform redirection in Rails routes as follows:

Version ≥ 4.0

```
get '/stories', to: redirect('/posts')
```

Version < 4.0

```
match "/abc" => redirect("http://example.com/abc")
```

You can also redirect all unknown routes to a given path:

Version ≥ 4.0

```
match '*path' => redirect('/'), via: :get
# or
get '*path' => redirect('/')
```

Version < 4.0

```
match '*path' => redirect('/')
```

Section 2.12: Redirects and Wildcard Routes

If you want to provide a URL out of convenience for your user but map it directly to another one you're already using. Use a redirect:

```
# config/routes.rb
TestApp::Application.routes.draw do
  get 'courses/:course_name' => redirect('/courses/%{course_name}/lessons'), :as => "course"
end
```

Well, that got interesting fast. The basic principle here is to just use the `#redirect` method to send one route to another route. If your route is quite simple, it's a really straightforward method. But if you want to also send the original parameters, you need to do a bit of gymnastics by capturing the parameter inside `%{here}`. Note the single quotes around everything.

In the example above, we've also renamed the route for convenience by using an alias with the :as parameter. This lets us use that name in methods like the #_path helpers. Again, test out your `$ rake routes` with questions.

Section 2.13: Scope available locales

If your application is available in different languages, you usually show the current locale in the URL.

```
scope '/(:locale)', locale: /#{I18n.available_locales.join('|')}/ do
    root 'example#root'
    # other routes
end
```

Your root will be accessible via the locales defined in `I18n.available_locales`.

第2.14节：带有句点的URL参数

如果你想支持比ID数字更复杂的URL参数，当参数值包含句点时，解析器可能会出现问题。句点后面的任何内容都会被认为是格式（例如json、xml）。

你可以通过使用约束来扩展接受的输入，从而绕过这个限制。

例如，如果你想通过电子邮件地址在 URL 中引用用户记录：

```
resources :users, constraints: { id: /.*/ }
```


Section 2.14: URL params with a period

If you want to support a url parameter more complex than an id number, you may run into trouble with the parser if the value contains a period. Anything following a period will be assumed to be a format (i.e. json, xml).

You can work around this limitation by using a constraint to *broaden* the accepted input.

For example, if you want to reference a user record by email address in the url:

```
resources :users, constraints: { id: /.*/ }
```

第3章：ActiveRecord

第3.1节：通过生成器创建模型

Ruby on Rails 提供了一个`model`生成器，你可以用它来创建 ActiveRecord 模型。只需使用`rails generate model`并提供模型名称。

```
$ rails g model user
```

除了在app/models中的模型文件外，生成器还会创建：

- 在 test/models/user_test.rb中的测试文件
- 在 test/fixtures/users.yml中的测试数据
- 在 db/migrate/XXX_create_users.rb中的数据库迁移文件

你也可以在生成模型时同时生成一些字段。

```
$ rails g model user email:string sign_in_count:integer birthday:date
```

这将在您的数据库中创建 email、sign_in_count 和 birthday 列，并设置相应的数据类型。

第3.2节：回调简介

回调是一种方法，在对象生命周期的特定时刻被调用（在创建、删除、更新、验证、保存或从数据库加载之前或之后）。

例如，假设你有一个在创建后30天内过期的列表。

实现这一点的一种方法如下：

```
class Listing < ApplicationRecord
  after_create :set_expiry_date

  private

  def set_expiry_date
expiry_date = Date.today + 30.days
    self.update_column(:expires_on, expiry_date)
  end
end
```

所有可用的回调方法如下，顺序与每个对象操作期间调用的顺序相同：

创建对象

- before_validation
- after_validation
- before_save
- around_save
- before_create
- around_create
- after_create
- after_save

Chapter 3: ActiveRecord

Section 3.1: Creating a Model via generator

Ruby on Rails provides a `model` generator you can use to create ActiveRecord models. Simply use `rails generate model` and provide the model name.

```
$ rails g model user
```

In addition to the model file in `app/models`, the generator will also create:

- the Test in `test/models/user_test.rb`
- the Fixtures in `test/fixtures/users.yml`
- the database Migration in `db/migrate/XXX_create_users.rb`

You can also generate some fields for the model when generating it.

```
$ rails g model user email:string sign_in_count:integer birthday:date
```

This will create the columns email, sign_in_count and birthday in your database, with the appropriate types.

Section 3.2: Introduction to Callbacks

A callback is a method that gets called at specific moments of an object's lifecycle (right before or after creation, deletion, update, validation, saving or loading from the database).

For instance, say you have a listing that expires within 30 days of creation.

One way to do that is like this:

```
class Listing < ApplicationRecord
  after_create :set_expiry_date

  private

  def set_expiry_date
    expiry_date = Date.today + 30.days
    self.update_column(:expires_on, expiry_date)
  end
end
```

All of the available methods for callbacks are as follows, in the same order that they are called during the operation of each object:

Creating an Object

- before_validation
- after_validation
- before_save
- around_save
- before_create
- around_create
- after_create
- after_save

- after_commit/after_rollback

更新对象

- 验证前
- 验证后
- 保存前
- 保存环绕
- 更新前
- 更新环绕
- 更新后
- 保存后
- after_commit/after_rollback

销毁对象

- 销毁前
- 销毁环绕
- 销毁后
- after_commit/after_rollback

注意： *after_save 在创建和更新时都会运行，但总是在更具体的回调 after_create 和 after_update 之后运行，无论宏调用的执行顺序如何。*

第3.3节：手动创建模型

虽然使用脚手架对于刚接触Rails或创建新应用来说快速且简单，但之后自己手动完成会更有用，以避免需要浏览脚手架生成的代码来精简它。
(移除未使用的部分等)。

创建模型可以简单到在app/models目录下创建一个文件。

在ActiveRecord中，最简单的模型是继承自ActiveRecord::Base的类。

```
class User < ActiveRecord::Base
end
```

模型文件存放在app/models/目录下，文件名对应类的单数形式：

```
# user
app/models/user.rb

# SomeModel
app/models/some_model.rb
```

该类将继承所有ActiveRecord的特性：查询方法、验证、回调等。

```
# 查找ID为1的User
User.find(1)
```

注意：确保对应模型的表存在。如果不存在，可以通过创建

Migration来创建该表

你可以通过以下命令在终端生成模型及其迁移文件

- after_commit/after_rollback

Updating an Object

- before_validation
- after_validation
- before_save
- around_save
- before_update
- around_update
- after_update
- after_save
- after_commit/after_rollback

Destroying an Object

- before_destroy
- around_destroy
- after_destroy
- after_commit/after_rollback

NOTE: *after_save runs both on create and update, but always after the more specific callbacks after_create and after_update, no matter the order in which the macro calls were executed.*

Section 3.3: Creating a Model manually

While using scaffolding is a fast and easy if you are new to Rails or you are creating a new application, later it can be useful just to do it on your own ato avoid the need to go through the scaffold-generated code to slim it down (remove unused parts, etc.).

Creating a model can be as simple as creating a file under `app/models`.

The most simple model, in `ActiveRecord`, is a class that extends `ActiveRecord::Base`.

```
class User < ActiveRecord::Base
end
```

Model files are stored in `app/models/`, and the file name corresponds to the singular name of the class:

```
# user
app/models/user.rb

# SomeModel
app/models/some_model.rb
```

The class will inherit all the ActiveRecord features: query methods, validations, callbacks, etc.

```
# Searches the User with ID 1
User.find(1)
```

Note: Make sure that the table for the corresponding model exists. If not, you can create the table by creating a Migration

You can generate a model and it's migration by terminal from the following command

```
rails g model column_name1:data_type1, column_name2:data_type2, ...
```

也可以通过以下命令为模型分配外键（关联关系）

```
rails g model column_name:data_type, model_name:references
```

第3.4节：手动测试你的模型

通过命令行界面测试你的Active Record模型很简单。进入终端中的app目录，输入rails console启动Rails控制台。从这里，你可以对数据库运行Active Record方法。

例如，如果你的数据库模式中Users表有一个name:string列和一个email:string列，你可以运行：

```
User.create name: "John", email: "john@example.com"
```

然后，要显示该记录，您可以运行：

```
User.find_by email: "john@example.com"
```

或者如果这是您的第一个或唯一的记录，您可以通过运行以下命令简单地获取第一条记录：

```
User.first
```

第3.5节：创建迁移

添加/删除现有表中的字段

通过运行以下命令创建迁移：

```
rails generate migration AddTitleToCategories title:string
```

这将创建一个迁移，向categories表中添加一个title列：

```
class AddTitleToCategories < ActiveRecord::Migration[5.0]
  def change
add_column :categories, :title, :string
  end
end
```

同样，你可以生成一个迁移来删除一列：rails generate migration RemoveTitleFromCategories title:string

这将创建一个迁移，从categories表中删除一个 title列：

```
class RemoveTitleFromCategories < ActiveRecord::Migration[5.0]
  def change
remove_column :categories, :title, :string
  end
end
```

严格来说，指定**type**（本例中为:**string**）对于删除列**不是必须的，但这很有帮助，**
因为它提供了回滚所需的信息。


```
rails g model column_name1:data_type1, column_name2:data_type2, ...
```

and can also assign foreign key(relationship) to the model by following command

```
rails g model column_name:data_type, model_name:references
```

Section 3.4: Manually Testing Your Models

Testing your Active Record models through your command line interface is simple. Navigate to the app directory in your terminal and type in `rails console` to start the Rails console. From here, you can run active record methods on your database.

For example, if you had a database schema with a Users table having a `name:string` column and `email:string`, you could run:

```
User.create name: "John", email: "john@example.com"
```

Then, to show that record, you could run:

```
User.find_by email: "john@example.com"
```

Or if this is your first or only record, you could simply get the first record by running:

```
User.first
```

Section 3.5: Creating A Migration

Add/remove fields in existing tables

Create a migration by running:

```
rails generate migration AddTitleToCategories title:string
```

This will create a migration that adds a `title` column to a `categories` table:

```
class AddTitleToCategories < ActiveRecord::Migration[5.0]
  def change
    add_column :categories, :title, :string
  end
end
```

Similarly, you can generate a migration to remove a column: `rails generate migration RemoveTitleFromCategories title:string`

This will create a migration that removes a `title` column from the `categories` table:

```
class RemoveTitleFromCategories < ActiveRecord::Migration[5.0]
  def change
    remove_column :categories, :title, :string
  end
end
```

While, strictly speaking, specifying **type** (`:string` in this case) is **not necessary** for removing a column, **it's helpful**, since it provides the information necessary for **rolling it back**.

创建表

通过运行以下命令创建迁移：

```
rails g CreateUsers name bio
```

Rails通过Create前缀识别创建表的意图，迁移名称的其余部分将用作表名。给出的示例生成如下内容：

```
class CreateUsers < ActiveRecord::Migration
  def change
create_table :users do |t|
t.字符串 :名称
t.string :bio
    end
  结束
结束
```

请注意，创建命令未指定列的类型，默认使用了string类型。

创建连接表

通过运行以下命令创建迁移：

```
rails g CreateJoinTableParticipation user:references group:references
```

Rails通过在迁移名称中找到JoinTable来检测创建连接表的意图。其余内容由你在名称后提供的字段名称决定。

```
class CreateJoinTableParticipation < ActiveRecord::Migration
  def change
create_join_table :users, :groups do |t|
      # t.index [:user_id, :group_id]
      # t.index [:group_id, :user_id]
    结束
  结束
end
```

取消注释必要的index语句并删除其余部分。

优先级

请注意，示例迁移名称CreateJoinTableParticipation符合表创建的规则：它有一个Create前缀。但它并没有生成一个简单的create_table。这是因为迁移生成器（source code）使用了以下列表的first match：

- (Add|Remove)<ignored>(To|From)<table_name>
- <ignored>JoinTable<ignored>
- Create<table_name>

第3.6节：使用迁移创建连接表

特别适用于has_and_belongs_to_many关联，你可以使用create_table方法手动创建连接表。假设你有两个模型Tags和Projects，并且你想通过

Create a table

Create a migration by running:

```
rails g CreateUsers name bio
```

Rails recognizes the intent to create a table from the `Create` prefix, the rest of the migration name will be used as a table name. The given example generates the following:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :bio
    end
  end
end
```

Notice that the creation command didn't specify types of columns and the default `string` was used.

Create a join table

Create a migration by running:

```
rails g CreateJoinTableParticipation user:references group:references
```

Rails detects the intent to create a join table by finding `JoinTable` in migration name. Everything else is determined from the names of the fields you give after the name.

```
class CreateJoinTableParticipation < ActiveRecord::Migration
  def change
    create_join_table :users, :groups do |t|
      # t.index [:user_id, :group_id]
      # t.index [:group_id, :user_id]
    end
  end
end
```

Uncomment the necessary `index` statements and delete the rest.

Precedence

Notice that the example migration name `CreateJoinTableParticipation` matches the rule for table creation: it has a `Create` prefix. But it did not generate a simple `create_table`. This is because migration generator (source code) uses a **first match** of the following list:

- (Add|Remove)<ignored>(To|From)<table_name>
- <ignored>JoinTable<ignored>
- Create<table_name>

Section 3.6: Create a Join Table using Migrations

Specially useful for `has_and_belongs_to_many` relation, you can manually create a join table using the `create_table` method. Suppose you have two models `Tags` and `Proyects`, and you'd like to associate them using a

has_and_belongs_to_many关联将它们关联起来。你需要一个连接表来关联这两个类的实例。

```
class CreateProjectsTagsJoinTableMigration < ActiveRecord::Migration
  def change
create_table :projects_tags, id: false do |t|
t.integer :project_id
t.integer :tag_id
    end
  结束
结束
```

表的实际名称需要遵循以下约定：按字母顺序排在前的模型必须放在前面。
项目（Project）排在标签（Tags）之前，因此表名为 projects_tags。

此外，由于此表的目的是关联两个模型实例之间的关系，因此该表中每条记录的实际ID并非必要。你可以通过传递 id: false 来指定这一点

最后，按照Rails的惯例，表名必须是各个模型名称的复合复数形式，但表中的列名必须是单数形式。

第3.7节：使用模型实例更新一行

假设你有一个User模型

```
class User < ActiveRecord::Base
end
```

现在要更新id = 1的用户的first_name和last_name，可以编写以下代码。

```
user = User.find(1)
user.update(first_name: 'Kashif', last_name: 'Liaqat')
```

调用update将尝试在单个事务中更新给定的属性，成功时返回**true**，失败时返回**false**。

has_and_belongs_to_many relation. You need a join table to associate instances of both classes.

```
class CreateProjectsTagsJoinTableMigration < ActiveRecord::Migration
  def change
    create_table :projects_tags, id: false do |t|
      t.integer :project_id
      t.integer :tag_id
    end
  end
end
```

The actual name of the table needs to follow this convention: the model which alphabetically precedes the other must go first. **P**roject preceds **T**ags so the name of the table is projects_tags.

Also since the purpose of this table is to route the association between the instances of two models, the actual id of every record in this table is not necessary. You specify this by passing `id: false`

Finally, as is convention in Rails, the table name must be the compound plural form of the individual models, but the column of the table must be in singular form.

Section 3.7: Using a model instance to update a row

Let's say you have a `User` model

```
class User < ActiveRecord::Base
end
```

Now to update the `first_name` and `last_name` of a user with `id = 1`, you can write the following code.

```
user = User.find(1)
user.update(first_name: 'Kashif', last_name: 'Liaqat')
```

Calling `update` will attempt to update the given attributes in a single transaction, returning **true** if successful and **false** if not.

第4章：视图

第4.1节：结构

由于 Rails 遵循 MVC 模式，视图是存放你操作的"模板"的地方。

假设你有一个控制器articles_controller.rb。对于这个控制器，你会在视图中有一个名为`app/views/articles`的文件夹：

```
app
|-- controllers
|   '-- articles_controller.rb
|
'-- views
'-- articles
|   |- index.html.erb
|   |- edit.html.erb
|   |- show.html.erb
|   |- new.html.erb
|   '- _partial_view.html.erb
|
'-- [...]
```

这种结构允许你为每个控制器拥有一个文件夹。当调用控制器中的某个操作时，
相应的视图将会自动渲染。

```
// articles_controller.rb
class ArticlesController < ActionController::Base
  def show
  end
end

// show.html.erb
<h1>我的显示视图</h1>
```

第4.2节：局部模板

局部模板（partials）是一种将渲染过程拆分成更易管理的块的方法。局部模板允许你将模板中的代码片段提取到单独的文件中，并且可以在整个模板中重复使用它们。

要*创建*一个局部模板，创建一个以下划线开头的新文件：`_form.html.erb`

要*渲染*一个局部模板作为视图的一部分，在视图中使用render方法：`<%= render "form" %>`

- 注意，渲染时省略下划线
- 如果局部模板位于不同文件夹，必须使用其路径进行渲染

要传递一个变量到局部模板作为局部变量，使用以下写法：

```
<%= render :partial => 'form', locals: { post: @post } %>
```

当你需要重用完全相同的代码时，局部模板也非常有用（DRY原则）。

例如，要重用<head>代码，创建一个名为_html_header.html.erb的局部模板，输入你的<head>代码

Chapter 4: Views

Section 4.1: Structure

As Rails follows the M**V**C pattern `Views` are where your "templates" are for your actions.

Let's say you have a controller `articles_controller.rb`. For this controller you would have a folder in views called `app/views/articles`:

```
app
|-- controllers
|   '-- articles_controller.rb
|
'-- views
'-- articles
|   |- index.html.erb
|   |- edit.html.erb
|   |- show.html.erb
|   |- new.html.erb
|   '- _partial_view.html.erb
|
'-- [...]
```

This structure allows you to have a folder for each controller. When calling an action in your controller the appropriate view will be rendered automatically.

```
// articles_controller.rb
class ArticlesController < ActionController::Base
  def show
  end
end

// show.html.erb
<h1>My show view</h1>
```

Section 4.2: Partials

Partial templates (partials) are a way of breaking the rendering process into more manageable chunks. Partials allow you to extract pieces of code from your templates to separate files and also reuse them throughout your templates.

To *create* a partial, create a new file that begins with an underscore: `_form.html.erb`

To *render* a partial as part of a view, use the render method within the view: `<%= render "form" %>`

- Note, the underscore is left out when rendering
- A partial has to be rendered using its path if located in a different folder

To *pass* a variable into the partial as a local variable, use this notation:

```
<%= render :partial => 'form', locals: { post: @post } %>
```

Partials are also useful when you need to *reuse* exactly the same code (**DRY philosophy**).

For example, to reuse `<head>` code, create a partial named `_html_header.html.erb`, enter your `<head>` code to be

通过以下方式重用，并在需要时渲染该局部：<%= render 'html_header' %>。

对象局部

响应to_partial_path的方法的对象也可以被渲染，例如<%= render @post %>。默认情况下，对于ActiveRecord模型，这通常是类似posts/post的路径，因此通过实际渲染@post，文件views/posts/_post.html.erb将被渲染。

一个名为post的局部变量将被自动分配。最终，<%= render @post %>是<%= render 'posts/post', post: @post %>的简写。

响应to_partial_path的方法的对象集合也可以被提供，例如<%= render @posts %>。每个项目将依次渲染。

全局局部

要创建一个可以在任何地方使用且无需引用其确切路径的全局局部，该局部必须位于views/application路径下。下面修改了之前的示例以说明此功能。

例如，这是一个全局局部的路径app/views/application/_html_header.html.erb：

要在任何地方渲染此全局局部，使用<%= render 'html_header' %>

第4.3节：AssetTagHelper

为了让 Rails 在大多数情况下自动且正确地链接资源（css/js/图片），你需要使用内置的辅助方法。（官方文档）

图片辅助方法

image_path

此方法返回位于 app/assets/images 中图片资源的路径。

```
image_path("edit.png") # => /assets/edit.png
```

image_url

此方法返回位于 app/assets/images 中图片资源的完整 URL。

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

image_tag

如果您想包含一个带有源设置的标签，请使用此辅助方法。

```
image_tag("icon.png") # => <img src="/assets/icon.png" alt="Icon" />
```

JavaScript 辅助方法

javascript_include_tag

如果您想在视图中包含一个 JavaScript 文件。

```
javascript_include_tag "application" # => <script src="/assets/application.js"></script>
```

javascript_path

reused, and render the partial whenever needed by: <%= render 'html_header' %>.

Object Partials

Objects that respond to to_partial_path can also be rendered, as in <%= render @post %>. By default, for ActiveRecord models, this will be something like posts/post, so by actually rendering @post, the file views/posts/_post.html.erb will be rendered.

A local named post will be automatically assigned. In the end, <%= render @post %> is a short hand for <%= render 'posts/post', post: @post %>.

Collections of objects that respond to to_partial_path can also be provided, such as <%= render @posts %>. Each item will be rendered consecutively.

Global Partials

To create a global partial that can be used anywhere without referencing its exact path, the partial has to be located in the views/application path. The previous example has been modified below to illustrate this feature.

For example, this is a path to a global partial app/views/application/_html_header.html.erb:

To render this global partial anywhere, use <%= render 'html_header' %>

Section 4.3: AssetTagHelper

To let rails automatically and correctly link assets (css/js/images) in most cases you want to use built in helpers. (Official documentation)

Image helpers

image_path

This returns the path to an image asset in app/assets/images.

```
image_path("edit.png") # => /assets/edit.png
```

image_url

This returns the full URL to an image asset in app/assets/images.

```
image_url("edit.png") # => http://www.example.com/assets/edit.png
```

image_tag

Use this helper if you want to include an -tag with the source set.

```
image_tag("icon.png") # => <img src="/assets/icon.png" alt="Icon" />
```

JavaScript helpers

javascript_include_tag

If you want to include a JavaScript-file in your view.

```
javascript_include_tag "application" # => <script src="/assets/application.js"></script>
```

javascript_path

此方法返回您的 JavaScript 文件路径。

```
javascript_path "application" # => /assets/application.js
```

javascript_url

这将返回您的JavaScript文件的完整URL。

```
javascript_url "application" # => http://www.example.com/assets/application.js
```

样式表辅助方法

stylesheet_link_tag

如果你想在视图中包含一个CSS文件。

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media="screen"
rel="stylesheet" />
```

stylesheet_path

返回你的样式表资源的路径。

```
stylesheet_path "application" # => /assets/application.css
```

stylesheet_url

返回你的样式表资源的完整URL。

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

示例用法

创建新的 Rails 应用时，您将自动拥有这两个辅助方法
app/views/layouts/application.html.erb

```
<%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

输出如下：

```
// CSS
<link rel="stylesheet" media="all" href="/assets/application.self-
e19d4b856cacba4f6fb0e5aa82a1ba9aa4ad616f0213a1259650b281d9cf6b20.css?body=1" data-turbolinks-
track="reload" />
// JavaScript
<script
src="/assets/application.self-619d9bf310b8eb258c67de7af745cafbf2a98f6d4c7bb6db8e1b00aed89eb0b1.js?b
ody=1" data-turbolinks-track="reload"></script>
```

第4.4节：替换视图中的HTML代码

如果你曾经想在运行时确定页面上要打印的HTML内容，那么Rails有一个非常好的解决方案。它有一个叫做content_for的功能，允许我们向Rails视图传递一个块。
请查看下面的示例，

声明content_for

This returns the path of your JavaScript-file.

```
javascript_path "application" # => /assets/application.js
```

javascript_url

This returns the full URL of your JavaScript-file.

```
javascript_url "application" # => http://www.example.com/assets/application.js
```

Stylesheet helpers

stylesheet_link_tag

If you want to include a CSS-file in your view.

```
stylesheet_link_tag "application" # => <link href="/assets/application.css" media="screen"
rel="stylesheet" />
```

stylesheet_path

This returns the path of you stylesheet asset.

```
stylesheet_path "application" # => /assets/application.css
```

stylesheet_url

This returns the full URL of you stylesheet asset.

```
stylesheet_url "application" # => http://www.example.com/assets/application.css
```

Example usage

When creating a new rails app you will automatically have two of these helpers in
app/views/layouts/application.html.erb

```
<%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track': 'reload' %>
<%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
```

This outputs:

```
// CSS
<link rel="stylesheet" media="all" href="/assets/application.self-
e19d4b856cacba4f6fb0e5aa82a1ba9aa4ad616f0213a1259650b281d9cf6b20.css?body=1" data-turbolinks-
track="reload" />
// JavaScript
<script
src="/assets/application.self-619d9bf310b8eb258c67de7af745cafbf2a98f6d4c7bb6db8e1b00aed89eb0b1.js?b
ody=1" data-turbolinks-track="reload"></script>
```

Section 4.4: Replace HTML code in Views

If you ever wanted to determine the html content to be printed on a page during run time then, rails has a very good solution for that. It has something called the **content_for** which allows us to pass a block to a rails view. Please check the below example,

Declare content_for

```
<div>
  <%= yield :header %>
</div>

<% content_for :header do %>
 <ul>
   <li>项目1</li>
   <li>项目2</li>
 </ul>
<% end %>
```

第4.5节：HAML——在视图中使用的另一种方式

HAML（HTML抽象标记语言）是一种优美且简洁的方式，用于描述和设计视图的HTML。HAML不使用开始和结束标签，而是通过缩进来表示页面的结构。基本上，如果某个内容应该放在另一个元素内，只需使用一个制表符缩进即可。制表符和空白在HAML中非常重要，因此请确保始终使用相同数量的制表符。

示例：

```
#myview.html.erb
<h1><%= @the_title %></h1>
<p>这是我的表单</p>
<%= render "form" %>
```

而在HAML中：

```
#myview.html.haml
%h1= @the_title
%p
这是我的表单
= render 'form'
```

你看，布局的结构比使用HTML和ERB更加清晰。

安装

只需使用以下命令安装该 gem

```
gem install haml
```

并将该 gem 添加到 Gemfile 中

```
gem "haml"
```

要使用 HAML 替代 HTML/ERB，只需将视图文件的扩展名从 `something.html.erb` 替换为 `something.html.haml`。

快速提示

常见元素如 div 可以用简短的方式书写

HTML

```
<div class="myclass">我的文本</div>
```



```
<div>
  <%= yield :header %>
</div>

<% content_for :header do %>
 <ul>
   <li>Line Item 1</li>
   <li>Line Item 2</li>
 </ul>
<% end %>
```

Section 4.5: HAML - an alternative way to use in your views

HAML (HTML abstraction markup language) is a beautiful and elegant way to describe and design the HTML of your views. Instead of opening- and closing tags, HAML uses indentation for the structure of your pages. Basically, if something should be placed within another element, you just indent it by using one tab stop. Tabs and white space are important in HAML, so be sure that you always use the same amount of tabs.

Examples:

```
#myview.html.erb
<h1><%= @the_title %></h1>
<p>This is my form</p>
<%= render "form" %>
```

And in HAML:

```
#myview.html.haml
%h1= @the_title
%p
    This is my form
= render 'form'
```

You see, the structure of the layout is much clearer than using HTML and ERB.

Installation

Just install the gem using

```
gem install haml
```

and add the gem to the Gemfile

```
gem "haml"
```

For using HAML instead of HTML/ERB, just replace the file extensions of your views from `something.html.erb` to `something.html.haml`.

Quick tipps

Common elements like divs can be written in a short way

HTML

```
<div class="myclass">My Text</div>
```

HAML

```
%div.myclass
```

HAML，简写

```
.myclass
```

属性

HTML

```
<p class="myclass" id="myid">我的段落</p>
```

HAML

```
%p{:class => "myclass", :id => "myid"} 我的段落
```

插入 Ruby 代码

你可以使用 = 和 - 符号插入 Ruby 代码。

```
= link_to "首页", home_path
```

以 = 开头的代码将被执行并嵌入文档中。

以-开头的代码将被执行，但不会插入到文档中。

完整文档

HAML 非常容易入门，但也非常复杂，因此我建议阅读文档。

HAML

```
%div.myclass
```

HAML, shorthand

```
.myclass
```

Attributes

HTML

```
<p class="myclass" id="myid">My paragraph</p>
```

HAML

```
%p{:class => "myclass", :id => "myid"} My paragraph
```

Inserting ruby code

You can insert ruby code with the = and - signs.

```
= link_to "Home", home_path
```

Code starting with **=** will be executed and embedded into the document.

Code starting with **-** will be executed, but not inserted into the document.

Full documentation

HAML is very easy to start with, but is also very complex, so that I'll recommend reading the documentation.

第5章：ActiveRecord 迁移

列类型	描述
:primary_key	主键
:string	较短的字符串数据类型。允许使用limit选项来限制最大字符数。
:text	较长的文本。允许使用limit选项来限制最大字节数。
:integer	整数。允许limit选项来限制最大字节数。
:bigint	更大的整数
:float	浮点数
:decimal	具有可变精度的小数。允许precision和scale选项。
:numeric	允许precision和scale选项。
:datetime	用于日期/时间的DateTime对象。
:time	时间对象，用于时间。
:date	日期对象，用于日期。
:binary	二进制数据。允许使用limit选项来限制最大字节数。
:boolean	布尔值

第5.1节：向表中添加多列

要向表中添加多列，在使用rails generate migration

命令时，将field:type对用空格分隔。

一般语法如下：

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

例如，下面的命令会向 users 表添加 name、salary 和 email 字段：

```
rails generate migration AddDetailsToUsers name:string salary:decimal email:string
```

该命令会生成如下迁移文件：

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :name, :string
    add_column :users, :salary, :decimal
    add_column :users, :email, :string
  end
end
```

第5.2节：向表中添加引用列

要向 users 表添加对 team 的引用，运行以下命令：

```
$ rails generate migration AddTeamRefToUsers team:references
```

这产生了以下迁移：

```
class AddTeamRefToUsers < ActiveRecord::Migration[5.0]
  def change
add_reference :users, :team, foreign_key: true
```


Chapter 5: ActiveRecord Migrations

Column type	Description
:primary_key	Primary key
:string	Shorter string datatype. Allows `limit` option for maximum number of characters.
:text	Longer amount of text. Allows `limit` option for maximum number of bytes.
:integer	Integer. Allows `limit` option for maximum number of bytes.
:bigint	Larger integer
:float	Float
:decimal	Decimal number with variable precision. Allows `precision` and `scale` options.
:numeric	Allows `precision` and `scale` options.
:datetime	DateTime object for dates/times.
:time	Time object for times.
:date	Date object for dates.
:binary	Binary data. Allows `limit` option for maximum number of bytes.
:boolean	Boolean

Section 5.1: Adding multiple columns to a table

To add multiple columns to a table, separate `field:type` pairs with spaces when using `rails generate migration` command.

The general syntax is:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

For example, the following will add `name`, `salary` and `email` fields to the `users` table:

```
rails generate migration AddDetailsToUsers name:string salary:decimal email:string
```

Which generates the following migration:

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
    add_column :users, :salary, :decimal
    add_column :users, :email, :string
  end
end
```

Section 5.2: Add a reference column to a table

To add a reference to a `team` to the `users` table, run this command:

```
$ rails generate migration AddTeamRefToUsers team:references
```

This generates the following migration:

```
class AddTeamRefToUsers < ActiveRecord::Migration[5.0]
  def change
    add_reference :users, :team, foreign_key: true
```

```
  结束
结束
```

该迁移将在 users 表中创建一个 team_id 列。

如果你想在新增的列上添加合适的 index 和 foreign_key，请将命令改为 rails generate migration AddTeamRefToUsers team:references:index。这样会生成以下迁移：

```
class AddTeamRefToUsers < ActiveRecord::Migration
  def change
add_reference :users, :team, index: true
    add_foreign_key :users, :teams
  结束
结束
```

如果你想给引用列命名为非 Rails 自动生成的名称，请在迁移中添加以下内容：（例如：你可能想在 Post 表中将创建 Post 的 User 称为 Author）

```
class AddAuthorRefToPosts < ActiveRecord::Migration
  def change
add_reference :posts, :author, references: :users, index: true
  end
end
```

第5.3节：回滚迁移

要回滚到最新的迁移，可以通过调用change方法的回滚，或者运行down方法。运行命令：

版本 < 5.0

```
rake db:rollback
```

版本 ≥ 5.0

```
rails db:rollback
```

回滚最近的3次迁移

版本 < 5.0

```
rake db:rollback STEP=3
```

版本 ≥ 5.0

```
rails db:rollback STEP=3
```

STEP 指定要回滚的迁移次数。

回滚所有迁移

版本 < 5.0

```
rake db:rollback VERSION=0
```

版本 ≥ 5.0

```
rails db:rollback VERSION=0
```

第5.4节：添加带索引的新列

要向users表中添加一个新的indexed列email，运行以下命令：

```
rails generate migration AddEmailToUsers email:string:index
```

这将生成以下迁移文件：


```
  end
end
```

That migration will create a `team_id` column in the `users` table.

If you want to add an appropriate `index` and `foreign_key` on the added column, change the command to `rails generate migration AddTeamRefToUsers team:references:index`. This will generate the following migration:

```
class AddTeamRefToUsers < ActiveRecord::Migration
  def change
    add_reference :users, :team, index: true
    add_foreign_key :users, :teams
  end
end
```

If you want to name your reference column other than what Rails auto generates, add the following to your migration: (E.g.: You might want to call the `User` who created the `Post` as `Author` in the `Post` table)

```
class AddAuthorRefToPosts < ActiveRecord::Migration
  def change
    add_reference :posts, :author, references: :users, index: true
  end
end
```

Section 5.3: Rollback migrations

To `rollback` the latest migration, either by reverting the `change` method or by running the `down` method. Run command:

Version < 5.0

```
rake db:rollback
```

Version ≥ 5.0

```
rails db:rollback
```

Rollback the last 3 migrations

Version < 5.0

```
rake db:rollback STEP=3
```

Version ≥ 5.0

```
rails db:rollback STEP=3
```

STEP provide the number of migrations to revert.

Rollback all migrations

Version < 5.0

```
rake db:rollback VERSION=0
```

Version ≥ 5.0

```
rails db:rollback VERSION=0
```

Section 5.4: Add a new column with an index

To add a new *indexed* column `email` to the `users` table, run the command:

```
rails generate migration AddEmailToUsers email:string:index
```

This will generate the following migration:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :email, :string
    add_index :users, :email
  结束
结束
```

第5.5节：运行特定迁移

要向上或向下运行特定迁移，使用 db:migrate:up或 db:migrate:down。

向上运行特定迁移：

版本 < 5.0

```
rake db:migrate:up VERSION=20090408054555
```

版本 ≥ 5.0

```
rails db:migrate:up VERSION=20090408054555
```

回滚指定的迁移：

版本 < 5.0

```
rake db:migrate:down VERSION=20090408054555
```

版本 ≥ 5.0

```
rails db:migrate:down VERSION=20090408054555
```

上述命令中的版本号是迁移文件名中的数字前缀。例如，要迁移到迁移文件20160515085959_add_name_to_users.rb，您应使用20160515085959作为版本号。

第5.6节：重新执行迁移

您可以使用redo命令先回滚然后再迁移。这基本上是rollback和migrate任务的快捷组合。

运行命令：

版本 < 5.0

```
rake db:migrate:redo
```

版本 ≥ 5.0

```
rails db:migrate:redo
```

您可以使用STEP参数回退多个版本。

例如，要回退3次迁移：

版本 < 5.0

```
rake db:migrate:redo STEP=3
```

版本 ≥ 5.0

```
rails db:migrate:redo STEP=3
```

第5.7节：向表中添加新列

要向users表中添加新列name，运行以下命令：


```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email
  end
end
```

Section 5.5: Run specific migration

To run a specific migration up or down, use `db:migrate:up` or `db:migrate:down`.

Up a specific migration:

Version < 5.0

```
rake db:migrate:up VERSION=20090408054555
```

Version ≥ 5.0

```
rails db:migrate:up VERSION=20090408054555
```

Down a specific migration:

Version < 5.0

```
rake db:migrate:down VERSION=20090408054555
```

Version ≥ 5.0

```
rails db:migrate:down VERSION=20090408054555
```

The version number in the above commands is the numeric prefix in the migration's filename. For example, to migrate to the migration `20160515085959_add_name_to_users.rb`, you would use `20160515085959` as the version number.

Section 5.6: Redo migrations

You can rollback and then migrate again using the `redo` command. This is basically a shortcut that combines `rollback` and `migrate` tasks.

Run command:

Version < 5.0

```
rake db:migrate:redo
```

Version ≥ 5.0

```
rails db:migrate:redo
```

You can use the `STEP` parameter to go back more than one version.

For example, to go back 3 migrations:

Version < 5.0

```
rake db:migrate:redo STEP=3
```

Version ≥ 5.0

```
rails db:migrate:redo STEP=3
```

Section 5.7: Add a new column to a table

To add a new column `name` to the `users` table, run the command:

```
rails generate migration AddNameToUsers name
```

这产生了以下迁移：

```
class AddNameToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :name, :string
  end
end
```

当迁移名称的形式为AddXXXToTABLE_NAME，后跟带数据类型的列列表时，生成的迁移将包含相应的add_column语句。

第5.8节：从表中删除现有列

要从users表中删除现有列name，运行以下命令：

```
rails generate migration RemoveNameFromUsers name:string
```

这将生成以下迁移文件：

```
class RemoveNameFromUsers < ActiveRecord::Migration[5.0]
  def change
remove_column :users, :name, :string
  end
end
```

当迁移名称的形式为RemoveXXXFromYYY，后面跟着带数据类型的列列表时，生成的迁移将包含相应的remove_column语句。

虽然不要求将数据类型（例如:string）作为remove_column的参数指定，但强烈建议这样做。如果未指定数据类型，则迁移将不可逆。

第5.9节：添加带默认值的列

下面的示例向users表添加一个名为admin的列，并将该列的默认值设为false。

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :admin, :boolean, default: false
  end
end
```

带默认值的迁移在大型表（例如PostgreSQL）中可能需要较长时间。这是因为每一行都必须用新添加列的默认值进行更新。为避免这种情况（并减少部署期间的停机时间），你可以将迁移拆分为三个步骤：

1. 添加一个类似上述的add_column迁移，但不设置默认值
2. 在应用运行时，通过 rake 任务或控制台部署并更新该列。确保您的应用已经为新建/更新的行写入该列的数据。
3. 添加另一个change_column迁移，然后将该列的默认值更改为所需的默认值值


```
rails generate migration AddNameToUsers name
```

This generates the following migration:

```
class AddNameToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :name, :string
  end
end
```

When the migration name is of the form `AddXXXToTABLE_NAME` followed by list of columns with data types, the generated migration will contain the appropriate `add_column` statements.

Section 5.8: Remove an existing column from a table

To remove existing column `name` from `users` table, run the command:

```
rails generate migration RemoveNameFromUsers name:string
```

This will generate the following migration:

```
class RemoveNameFromUsers < ActiveRecord::Migration[5.0]
  def change
    remove_column :users, :name, :string
  end
end
```

When the migration name is of the form `RemoveXXXFromYYY` followed by list of columns with data types then the generated migration will contain the appropriate `remove_column` statements.

While it's not required to specify the data type (e.g. `:string`) as a parameter to `remove_column`, it is highly recommended. If the data type is *not* specified, then the migration will not be reversible.

Section 5.9: Add column with default value

The following example adds a column `admin` to the `users` table, and gives that column the default value `false`.

```
class AddDetailsToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :admin, :boolean, default: false
  end
end
```

Migrations with defaults might take a long time in large tables with for example PostgreSQL. This is because each row will have to be updated with the default value for the newly added column. To circumvent this (and reduce downtime during deployments), you can split your migration into three steps:

1. Add a `add_column`-migration similar to the one above, but set no default
2. Deploy and update the column in a rake task or on the console while your app is running. Make sure your application already writes data to that colum for new/updated rows.
3. Add another `change_column` migration, which then changes the default of that column to the desired default value

第5.10节：在不同环境中运行迁移

要在测试环境中运行迁移，请执行以下 shell 命令：

```
rake db:migrate RAILS_ENV=test
```

版本 ≥ 5.0

从 Rails 5.0 开始，您可以使用rails代替rake：

```
rails db:migrate RAILS_ENV=test
```

第5.11节：创建新表

要创建一个包含name和salary列的新users表，请运行以下命令：

```
rails generate migration CreateUsers name:string salary:decimal
```

这将生成以下迁移文件：

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
create_table :users do |t|
t.字符串 :名称
t.decimal :salary
    end
  结束
结束
```

当迁移名称的形式为CreateXXX，后跟带数据类型的列列表时，将生成一个迁移，
该迁移会创建名为XXX的表，并包含列出的字段。

第5.12节：运行迁移

运行命令：

版本 < 5.0

```
rake db:migrate
```

版本 ≥ 5.0

```
rails db:migrate
```

指定目标版本将运行所需的迁移（up、down、change），直到达到指定的
版本。这里，版本号是迁移文件名中的数字前缀。

版本 < 5.0

```
rake db:migrate VERSION=20080906120000
```

版本 ≥ 5.0

```
rails db:migrate VERSION=20080906120000
```

第5.13节：更改现有列的类型

要通过迁移修改Rails中的现有列，请运行以下命令：

```
rails g migration change_column_in_table
```


Section 5.10: Running migrations in different environments

To run migrations in the `test` environment, run this shell command:

```
rake db:migrate RAILS_ENV=test
```

Version ≥ 5.0

Starting in Rails 5.0, you can use `rails` instead of `rake`:

```
rails db:migrate RAILS_ENV=test
```

Section 5.11: Create a new table

To create a new `users` table with the columns `name` and `salary`, run the command:

```
rails generate migration CreateUsers name:string salary:decimal
```

This will generate the following migration:

```
class CreateUsers < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      t.string :name
      t.decimal :salary
    end
  end
end
```

When the migration name is of the form `CreateXXX` followed by list of columns with data types, then a migration will be generated that creates the table `XXX` with the listed columns.

Section 5.12: Running migrations

Run command:

Version < 5.0

```
rake db:migrate
```

Version ≥ 5.0

```
rails db:migrate
```

Specifying target version will run the required migrations (up, down, change) until it has reached the specified version. Here, `version number` is the numerical prefix on the migration's filename.

Version < 5.0

```
rake db:migrate VERSION=20080906120000
```

Version ≥ 5.0

```
rails db:migrate VERSION=20080906120000
```

Section 5.13: Change an existing column's type

To modify an existing column in Rails with a migration, run the following command:

```
rails g migration change_column_in_table
```

这将在 db/migration 目录下创建一个新的迁移文件（如果该目录尚不存在），该文件名以时间戳和迁移文件名为前缀，内容如下：

```
def change
change_column (:table_name, :column_name, :new_type)
end
```

Rails 指南 – 修改列

更长但更安全的方法

上述代码会阻止用户回滚迁移。你可以通过将 `change` 方法拆分为单独的 `up` 和 `down` 方法来避免这个问题：

```
def up
change_column :my_table, :my_column, :new_type
end

def down
change_column :my_table, :my_column, :old_type
end
```

第5.14节：创建hstore列

`Hstore` 列可以用于存储设置。在启用

扩展后，它们可用于PostgreSQL数据库。

```
class CreatePages < ActiveRecord::Migration[5.0]
  def change
create_table :pages do |t|
enable_extension 'hstore' unless extension_enabled?('hstore')
t.hstore :settings
t.timestamps
    结束
  结束
end
```

第5.15节：创建连接表

要创建 students 和 courses 之间的连接表，请运行命令：

```
$ rails g migration CreateJoinTableStudentCourse student course
```

这将生成以下迁移文件：

```
class CreateJoinTableStudentCourse < ActiveRecord::Migration[5.0]
  def change
create_join_table :students, :courses do |t|
      # t.index [:student_id, :course_id]
      # t.index [:course_id, :student_id]
    结束
  结束
end
```


This will create a new migration file in `db/migration` directory (if it doesn't exist already), which will contain the file prefixed with timestamp and migration file name which contains the below content:

```
def change
  change_column(:table_name, :column_name, :new_type)
end
```

Rails Guide – Changing Columns

A longer but safer method

The above code prevents the user from ever rolling back the migration. You can avoid this problem by splitting the `change` method into separate `up` and `down` methods:

```
def up
  change_column :my_table, :my_column, :new_type
end

def down
  change_column :my_table, :my_column, :old_type
end
```

Section 5.14: Create a hstore column

`Hstore` columns can be useful to store settings. They are available in PostgreSQL databases after you enabled the extension.

```
class CreatePages < ActiveRecord::Migration[5.0]
  def change
    create_table :pages do |t|
      enable_extension 'hstore' unless extension_enabled?('hstore')
      t.hstore :settings
      t.timestamps
    end
  end
end
```

Section 5.15: Create a join table

To create a join table between `students` and `courses`, run the command:

```
$ rails g migration CreateJoinTableStudentCourse student course
```

This will generate the following migration:

```
class CreateJoinTableStudentCourse < ActiveRecord::Migration[5.0]
  def change
    create_join_table :students, :courses do |t|
      # t.index [:student_id, :course_id]
      # t.index [:course_id, :student_id]
    end
  end
end
```

第5.16节：添加自引用

自引用对于构建层级树结构非常有用。这可以通过迁移中的add_reference来实现。

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
add_reference :pages, :pages
  end
end
```

外键列将是pages_id。如果你想自定义外键列名，必须先创建该列，然后再添加引用。

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
add_column :pages, :parent_id, :integer, null: true, index: true
    add_foreign_key :pages, :pages, column: :parent_id
  结束
结束
```

第5.17节：创建数组列

PostgreSQL 支持数组列。Rails 会自动将 PostgreSQL 数组转换为 Ruby 数组，反之亦然。

创建一个带有数组列的表：

```
create_table :products do |t|
t.string :name
t.text :colors, array: true, 默认值: []
end
```

向现有表中添加数组列：

```
add_column :products, :colors, array: true, 默认值: []
```

为数组列添加索引：

```
add_index :products, :colors, 使用: 'gin'
```

第 5.18 节：向表中添加唯一列

要向users表中添加一个新的unique列email，请运行以下命令：

```
rails generate migration AddEmailToUsers email:string:uniq
```

这将创建以下迁移：

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :email, :string
    add_index :users, :email, unique: true
  end
end
```


Section 5.16: Add a self reference

A self reference can be useful to build a hierarchical tree. This can be achieved with `add_reference` in a migration.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_reference :pages, :pages
  end
end
```

The foreign key column will be `pages_id`. If you want to decide about the foreign key column name, you have to create the column first and add the reference after.

```
class AddParentPages < ActiveRecord::Migration[5.0]
  def change
    add_column :pages, :parent_id, :integer, null: true, index: true
    add_foreign_key :pages, :pages, column: :parent_id
  end
end
```

Section 5.17: Create an array column

An `array` column is supported by PostgreSQL. Rails will automatically convert a PostgreSQL array to a Ruby array, and vice-versa.

Create a table with an `array` column:

```
create_table :products do |t|
  t.string :name
  t.text :colors, array: true, default: []
end
```

Add an `array` column to an existing table:

```
add_column :products, :colors, array: true, default: []
```

Add an index for an `array` column:

```
add_index :products, :colors, using: 'gin'
```

Section 5.18: Add an unique column to a table

To add a new *unique* column `email` to `users`, run the following command:

```
rails generate migration AddEmailToUsers email:string:uniq
```

This will create the following migration:

```
class AddEmailToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :email, :string
    add_index :users, :email, unique: true
  end
end
```

第5.19节：检查迁移状态

我们可以通过运行以下命令来检查迁移状态

版本 ≥ 3.0 版本 < 5.0

```
rake db:migrate:status
```

版本 ≥ 5.0

```
rails db:migrate:status
```

输出将如下所示：

```
Status   Migration ID    Migration Name
--------------------------------------------------
up     20140711185212  Create documentation pages
up     20140724111844  Create nifty attachments table
up     20140724114255  Create documentation screenshots
up     20160213170731  Create owners
up     20160218214551  创建用户
up     20160221162159  ********** 无文件 **********
up     20160222231219  ********** 无文件 **********
```

在状态字段中，up 表示迁移已执行，down 表示需要执行迁移。

第5.20节：更改表

如果你创建了一个模式错误的表，那么更改列及其
属性的最简单方法是change_table。请查看以下示例：

```
change_table :orders do |t|
t.remove :ordered_at # 删除列 ordered_at
t.string :skew_number # 添加新列
t.index  :skew_number # 创建索引
t.rename :location, :state # 将 location 列重命名为 state
end
```

上述迁移更改了表orders。以下是对更改的逐行说明：

1. t.remove :ordered_at 从表 orders 中移除列 ordered_at。
2. t.string :skew_number 在 orders 表中添加一个名为 skew_number 的字符串类型新列。
3. t.index :skew_number 在 orders 表的 skew_number 列上添加索引。
4. t.rename :location, :state 将 orders 表中的 location 列重命名为 state。

第5.21节：向现有数据添加 NOT NULL 约束

假设你想向 users 表添加一个外键 company_id，并且想对其添加 NOT NULL 约束。
如果 users 表中已经有数据，你需要分多步完成此操作。

```
class AddCompanyIdToUsers < ActiveRecord::Migration
  def up
    # 添加允许为 NULL 的列
add_column :users, :company_id, :integer

    # 确保每一行都有值
User.find_each do |user|
      # 根据您的业务逻辑查找用户对应的公司记录
```


Section 5.19: Checking migration status

We can check the status of migrations by running

Version ≥ 3.0 Version < 5.0

```
rake db:migrate:status
```

Version ≥ 5.0

```
rails db:migrate:status
```

The output will look like this:

```
Status   Migration ID    Migration Name
--------------------------------------------------
up     20140711185212  Create documentation pages
up     20140724111844  Create nifty attachments table
up     20140724114255  Create documentation screenshots
up     20160213170731  Create owners
up     20160218214551  Create users
up     20160221162159  ********** NO FILE **********
up     20160222231219  ********** NO FILE **********
```

Under the status field, up means the migration has been run and down means that we need to run the migration.

Section 5.20: Changing Tables

If you have created a table with some wrong schema, then the easiest way to change the columns and their properties is change_table. Review the following example:

```
change_table :orders do |t|
  t.remove :ordered_at # removes column ordered_at
  t.string :skew_number # adds a new column
  t.index  :skew_number #creates an index
  t.rename :location, :state #renames location column to state
end
```

The above migration changes a table orders. Here is a line-by-line description of the changes:

1. t.remove :ordered_at removes the column ordered_at from the table orders.
2. t.string :skew_number adds a new string-type column named skew_number in the orders table.
3. t.index :skew_number adds an index on the skew_number column in the orders table.
4. t.rename :location, :state renames the location column in the orders table to state.

Section 5.21: Adding a NOT NULL constraint to existing data

Say you want to add a foreign key company_id to the users table, and you want to have a NOT NULL constraint on it. If you already have data in users, you will have to do this in multiple steps.

```
class AddCompanyIdToUsers < ActiveRecord::Migration
  def up
    # add the column with NULL allowed
    add_column :users, :company_id, :integer

    # make sure every row has a value
    User.find_each do |user|
      # find the appropriate company record for the user
```

```
      # 根据您的业务逻辑
company = Company.first
      user.update!(company_id: company.id)
    end

    # 添加 NOT NULL 约束
change_column_null :users, :company_id, false
  end

  # 操作数据的迁移必须使用 up/down 方法，而不是 change
  def down
remove_column :users, :company_id
  end
end
```

第5.22节：禁止空值

要禁止表列中的null值，请在迁移中添加:null参数，如下所示：

```
class AddPriceToProducts < ActiveRecord::Migration
  def change
add_column :products, :float, null: false
  end
end
```



```
      # according to your business logic
      company = Company.first
      user.update!(company_id: company.id)
    end

    # add NOT NULL constraint
    change_column_null :users, :company_id, false
  end

  # Migrations that manipulate data must use up/down instead of change
  def down
    remove_column :users, :company_id
  end
end
```

Section 5.22: Forbid null values

To forbid `null` values in your table columns, add the `:null` parameter to your migration, like this:

```
class AddPriceToProducts < ActiveRecord::Migration
  def change
    add_column :products, :float, null: false
  end
end
```

第6章：Rails最佳实践

第6.1节：胖模型，瘦控制器

“胖模型，瘦控制器”指的是MVC中模型（M）和控制器（C）理想的协作方式。也就是说，任何与响应无关的逻辑都应该放在模型中，最好是放在一个良好且可测试的方法里。与此同时，“瘦”控制器只是视图和模型之间的一个良好接口。

在实际操作中，这可能需要多种不同类型的重构，但归根结底只有一个理念：通过将任何与响应无关的逻辑移到模型（而不是控制器）中，不仅促进了代码的复用，还使得可以在请求上下文之外测试代码成为可能。

让'我们来看一个简单的例子。假设你有如下代码：

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

你可以将其改成这样：

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

然后，你可以将逻辑移到你的帖子模型中，代码可能如下所示：

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?',
timestamp) }
```

第6.2节：领域对象（不再是臃肿模型）

“臃肿模型，精简控制器”是一个很好的第一步，但当你的代码库开始增长时，这种方式扩展性不好。

让我们思考模型的单一职责。模型的单一职责是什么？是承载业务逻辑吗？是承载与响应无关的逻辑吗？

不是。它的职责是处理持久层及其抽象。

业务逻辑，以及任何与响应无关和与持久化无关的逻辑，都应该放在领域对象中。

领域对象是设计为在问题领域中只承担单一职责的类。让你的类“大声宣告它们的架构”，体现它们解决的问题。

在实践中，你应该努力实现精简模型、精简视图和精简控制器。你的解决方案架构不应受你选择的框架影响。

例如

假设你是一个通过Stripe向客户收取固定15%佣金的市场平台。如果你收取固定15%的佣金，这意味着你的佣金会根据订单金额而变化，因为Stripe

Chapter 6: Rails Best Practices

Section 6.1: Fat Model, Skinny Controller

“Fat Model, Skinny Controller” refers to how the M and C parts of MVC ideally work together. Namely, any non-response-related logic should go in the model, ideally in a nice, testable method. Meanwhile, the “skinny” controller is simply a nice interface between the view and model.

In practice, this can require a range of different types of refactoring, but it all comes down to one idea: by moving any logic that isn't about the response to the model (instead of the controller), not only have you promoted reuse where possible but you've also made it possible to test your code outside of the context of a request.

Let's look at a simple example. Say you have code like this:

```
def index
  @published_posts = Post.where('published_at <= ?', Time.now)
  @unpublished_posts = Post.where('published_at IS NULL OR published_at > ?', Time.now)
end
```

You can change it to this:

```
def index
  @published_posts = Post.published
  @unpublished_posts = Post.unpublished
end
```

Then, you can move the logic to your post model, where it might look like this:

```
scope :published, ->(timestamp = Time.now) { where('published_at <= ?', timestamp) }
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?',
timestamp) }
```

Section 6.2: Domain Objects (No More Fat Models)

"Fat Model, Skinny Controller" is a very good first step, but it doesn't scale well once your codebase starts to grow.

Let's think on the Single Responsibility of models. What is the single responsibility of models? Is it to hold business logic? Is it to hold non-response-related logic?

No. Its responsibility is to handle the persistence layer and its abstraction.

Business logic, as well as any non-response-related logic and non-persistence-related logic, should go in domain objects.

Domain objects are classes designed to have only one responsibility in the domain of the problem. Let your classes "Scream Their Architecture" for the problems they solve.

In practice, you should strive towards skinny models, skinny views and skinny controllers. The architecture of your solution shouldn't be influenced by the framework you're choosing.

For example

Let's say you're a marketplace which charges a fixed 15% commission to your customers via Stripe. If you charge a fixed 15% commission, that means that your commission changes depending on the order's amount because Stripe

收费 2.9% + 30¢。

您收取的佣金金额应为：amount*0.15 - (amount*0.029 + 0.30)。

不要在模型中编写此逻辑：

```
# app/models/order.rb
class Order < ActiveRecord::Base
  SERVICE_COMMISSION = 0.15
STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  ...

  def commission
amount*SERVICE_COMMISSION - stripe_commission
  end

  private

  def stripe_commission
amount*STRIPE_PERCENTAGE_COMMISSION + STRIPE_FIXED_COMMISSION
  结束
结束
```

一旦你集成了新的支付方式，就无法在这个

模型中扩展此功能。

此外，一旦你开始集成更多的业务逻辑，你的订单对象将开始失去内聚性。

优先使用领域对象，将佣金的计算完全从订单持久化的责任中抽象出来：

```
# app/models/order.rb
class Order < ActiveRecord::Base
  …
  # 不涉及佣金计算
end

# lib/commission.rb
class Commission
  SERVICE_COMMISSION = 0.15

  def self.calculate(支付方式, 模型)
    模型.金额*服务佣金 - 支付佣金(支付方式, 模型)
  end

  private

  def self.payment_commission(payment_method, model)
    # 有更好的方法来实现静态注册表，
    # 这里仅作示例说明。
    Object.const_get("#{payment_method}Commission").calculate(model)
  end
end

# lib/stripe_commission.rb
class StripeCommission
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30
```


charges 2.9% + 30¢.

The amount you charge as commission should be: `amount*0.15 - (amount*0.029 + 0.30)`.

Don't write this logic in the model:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  SERVICE_COMMISSION = 0.15
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30

  ...

  def commission
    amount*SERVICE_COMMISSION - stripe_commission
  end

  private

  def stripe_commission
    amount*STRIPE_PERCENTAGE_COMMISSION + STRIPE_FIXED_COMMISSION
  end
end
```

As soon as you integrate with a new payment method, you won't be able to scale this functionality inside this model.

Also, as soon as you start to integrate more business logic, your `Order` object will start to lose cohesion.

Prefer domain objects, with the calculation of the commission completely abstracted from the responsibility of persisting orders:

```
# app/models/order.rb
class Order < ActiveRecord::Base
  ...
  # No reference to commission calculation
end

# lib/commission.rb
class Commission
  SERVICE_COMMISSION = 0.15

  def self.calculate(payment_method, model)
    model.amount*SERVICE_COMMISSION - payment_commission(payment_method, model)
  end

  private

  def self.payment_commission(payment_method, model)
    # There are better ways to implement a static registry,
    # this is only for illustration purposes.
    Object.const_get("#{payment_method}Commission").calculate(model)
  end
end

# lib/stripe_commission.rb
class StripeCommission
  STRIPE_PERCENTAGE_COMMISSION = 0.029
  STRIPE_FIXED_COMMISSION = 0.30
```

```
  def self.calculate(model)
    model.amount*STRIPE_PERCENTAGE_COMMISSION
      + STRIPE_PERCENTAGE_COMMISSION
  结束
结束

# app/controllers/orders_controller.rb
class OrdersController < ApplicationController
  def create
    @order = Order.new(order_params)
    @order.commission = Commission.calculate("Stripe", @order)
    ...
  结束
结束
```

使用领域对象具有以下架构优势：

- 它非常容易进行单元测试，因为不需要使用夹具或工厂来实例化带有逻辑的对象。
- 适用于所有接受消息amount的对象。
- 保持每个领域对象小巧，职责明确，内聚性更高。
- 通过添加，而非修改，轻松扩展新的支付方式。
- 避免了在每个Ruby on Rails应用中User对象不断膨胀的趋势。

我个人喜欢将领域对象放在lib目录下。如果这样做，记得将其添加到autoload_paths中：

```
# config/application.rb
config.autoload_paths << Rails.root.join('lib')
```

你也可以选择让领域对象更具动作导向，遵循命令/查询模式。在这种情况下，将这些对象放在app/commands目录可能更合适，因为所有app子目录都会自动添加到自动加载路径中。

第6.3节：注意default_scope

ActiveRecord 包含default_scope，用于默认自动限定模型的作用域。

```
class Post
default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

上述代码将在你对模型执行任何查询时，只返回已发布的帖子。

```
Post.all # 只会列出已发布的帖子
```

这个作用域看似无害，但实际上有多个你可能不希望出现的隐藏副作用。

default_scope 和 order

由于你在default_scope中声明了order，调用Post的order时，会作为附加排序条件添加，而不是覆盖默认排序。

```
Post.order(updated_at: :desc)
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at" DESC,
"posts"."updated_at" DESC
```

这可能不是你想要的行为；你可以通过先从作用域中排除order来覆盖它


```
  def self.calculate(model)
    model.amount*STRIPE_PERCENTAGE_COMMISSION
      + STRIPE_PERCENTAGE_COMMISSION
  end
end

# app/controllers/orders_controller.rb
class OrdersController < ApplicationController
  def create
    @order = Order.new(order_params)
    @order.commission = Commission.calculate("Stripe", @order)
    ...
  end
end
```

Using domain objects has the following architectural advantages:

- it's extremely easy to unit test, as no fixtures or factories are required to instantiate the objects with the logic.
- works with everything that accepts the message `amount`.
- keeps each domain object small, with clearly defined responsibilities, and with higher cohesion.
- easily scales with new payment methods by addition, not modification.
- stops the tendency to have an ever-growing `User` object in each Ruby on Rails application.

I personally like to put domain objects in `lib`. If you do so, remember to add it to `autoload_paths`:

```
# config/application.rb
config.autoload_paths << Rails.root.join('lib')
```

You may also prefer to create domain objects more action-oriented, following the Command/Query pattern. In such case, putting these objects in `app/commands` might be a better place as all `app` subdirectories are automatically added to the autoload path.

Section 6.3: Beware of default_scope

ActiveRecord includes `default_scope`, to automatically scope a model by default.

```
class Post
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

The above code will serve posts which are already published when you perform any query on the model.

```
Post.all # will only list published posts
```

That scope, while innocuous-looking, has multiple hidden side-effect that you may not want.

`default_scope` and `order`

Since you declared an `order` in the `default_scope`, calling `order` on `Post` will be added as additional orders instead of overriding the default.

```
Post.order(updated_at: :desc)
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at" DESC,
"posts"."updated_at" DESC
```

This is probably not the behavior you wanted; you can override this by excluding the `order` from the scope first

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at" DESC
```

default_scope 和模型初始化

与其他任何ActiveRecord::Relation一样，default_scope会改变从中初始化的模型的默认状态。

在上面的例子中，Post默认设置了where(published: true)，因此从Post创建的新模型也会
默认设置该条件。

```
Post.new # => <Post published: true>
```

unscoped

default_scope名义上可以通过先调用unscoped来清除，但这也有副作用。例如，考虑一个STI模型：

```
class Post < Document
default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

默认情况下，对Post的查询会限定在包含'Post'的type列上。但unscoped会清除这一限制
以及你自己的default_scope，因此如果使用unscoped，你必须记得同时考虑这一点。

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

unscoped 和 模型关联

考虑Post和User之间的关系

```
class Post < ApplicationRecord
  belongs_to :user
default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

通过获取单个User，可以查看与其相关的帖子：

```
user = User.find(1)
user.posts
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' AND "posts"."user_id" = ? ORDER BY "posts"."created_at" DESC [["user_id", 1]]
```

但你想清除posts关联中的default_scope，所以你使用了unscoped

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

这会清除user_id条件以及default_scope。

一个使用default_scope的示例用例

尽管如此，在某些情况下使用default_scope是合理的。


```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at" DESC
```

default_scope and model initialization

As with any other ActiveRecord::Relation, default_scope will alter the default state of models initialized from it.

In the above example, Post has where(published: true) set by default, and so new models from Post will also have it set.

```
Post.new # => <Post published: true>
```

unscoped

default_scope can nominally be cleared by calling unscoped first, but this also has side-effects. Take, for example, an STI model:

```
class Post < Document
  default_scope ->{ where(published: true).order(created_at: :desc) }
end
```

By default, queries against Post will be scoped to type columns containing 'Post'. But unscoped will clear this along with your own default_scope, so if you use unscoped you have to remember to account for this as well.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

unscoped and Model Associations

Consider a relationship between Post and User

```
class Post < ApplicationRecord
  belongs_to :user
  default_scope ->{ where(published: true).order(created_at: :desc) }
end

class User < ApplicationRecord
  has_many :posts
end
```

By getting an individual User, you can see the posts related to it:

```
user = User.find(1)
user.posts
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' AND "posts"."user_id" = ? ORDER BY "posts"."created_at" DESC [["user_id", 1]]
```

But you want to clear the default_scope from the posts relation, so you use unscoped

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

This wipes out the user_id condition as well as the default_scope.

An example use-case for default_scope

Despite all of that, there are situations where using default_scope is justifiable.

考虑一个多租户系统，其中多个子域名由同一个应用程序提供服务，但数据是隔离的。实现这种隔离的一种方法是通过default_scope。在其他情况下的缺点在这里变成了优点。

```
class ApplicationRecord < ActiveRecord::Base
  def self.inherited(subclass)
    super

    return unless subclass.superclass == self
    return unless subclass.column_names.include? 'tenant_id'

subclass.class_eval do
default_scope ->{ where(tenant_id: Tenant.current_id) }
    结束
  结束
end
```

你所需要做的就是在请求早期将Tenant.current_id设置为某个值，任何包含tenant_id的表都会自动应用作用域，无需额外代码。实例化记录时会自动继承它们创建时的租户ID。

这个用例的重要之处在于作用域在每个请求中只设置一次，并且不会改变。你唯一需要使用unscoped的情况是像后台工作进程这类在请求作用域之外运行的特殊情况。

第6.4节：约定优于配置

在 Rails 中，你会看到用于数据库的控制器、视图和模型。

为了减少繁琐的配置，Rails 实现了一些规则来简化应用程序的开发。你可以定义自己的规则，但对于初学者（以及后续开发）来说，遵循 Rails 提供的约定是个不错的选择。

这些约定将加快开发速度，使代码简洁且易读，并且方便你在应用程序中轻松导航。

约定还降低了初学者的入门门槛。Rails 中有许多约定，初学者甚至不需要了解它们，只需在不知情的情况下受益。完全可以在不了解所有细节的情况下创建出色的应用程序。

例如

如果你有一个名为orders的数据库表，主键为id，那么对应的模型名为order，处理所有逻辑的控制器名为orders_controller。视图则根据不同的动作拆分：如果控制器有new和edit动作，那么也会有对应的new和edit视图。

例如

要创建一个应用程序，你只需运行rails new app_name。这样会生成大约70个文件和文件夹，构成你的 Rails 应用的基础设施和框架。

它包括：

- 用于存放模型（数据库层）、控制器和视图的文件夹
- 用于存放应用单元测试的文件夹
- 用于存放网页资源如 Javascript 和 CSS 文件的文件夹
- HTTP 400 响应的默认文件（例如文件未找到）

Consider a multi-tenant system where multiple subdomains are served from the same application but with isolated data. One way to achieve this isolation is through `default_scope`. The downsides in other cases become upsides here.

```
class ApplicationRecord < ActiveRecord::Base
  def self.inherited(subclass)
    super

    return unless subclass.superclass == self
    return unless subclass.column_names.include? 'tenant_id'

    subclass.class_eval do
      default_scope ->{ where(tenant_id: Tenant.current_id) }
    end
  end
end
```

All you need to do is set `Tenant.current_id` to something early in the request, and any table that contains `tenant_id` will automatically become scoped without any additional code. Instantiating records will automatically inherit the tenant id they were created under.

The important thing about this use-case is that the scope is set once per request, and it doesn't change. The only cases you will need `unscoped` here are special cases like background workers that run outside of a request scope.

Section 6.4: Convention Over Configuration

In Rails, you find yourself looking at *controllers, views, and models* for your database.

To reduce the need for heavy configuration, Rails implements rules to ease up working with the application. You may define your own rules but for the beginning (and for later on) it's a good idea to stick to conventions that Rails offers.

These conventions will speed up development, keep your code concise and readable, and allow you an easy navigation inside your application.

Conventions also lower the barriers to entry for beginners. There are so many conventions in Rails that a beginner doesn't even need to know about, but can just benefit from in ignorance. It's possible to create great applications without knowing why everything is the way it is.

For Example

If you have a database table called `orders` with the primary key `id`, the matching model is called `order` and the controller that handles all the logic is named `orders_controller`. The view is split in different actions: if the controller has a `new` and `edit` action, there is also a `new` and `edit` view.

For Example

To create an app you simply run `rails new app_name`. This will generate roughly 70 files and folders that comprise the infrastructure and foundation for your Rails app.

It includes:

- Folders to hold your models (database layer), controllers, and views
- Folders to hold unit tests for your application
- Folders to hold your web assets like Javascript and CSS files
- Default files for HTTP 400 responses (i.e. file not found)

- 以及其他许多内容

第6.5节：不要重复自己（DRY）

为了帮助保持代码整洁，Rails 遵循 DRY 原则。

这意味着尽可能重用代码，而不是在多个地方复制类似的代码（例如，使用局部视图）。这减少了错误，保持代码整洁，并且贯彻了"只写一次代码然后重用"的原则。更新代码时，只需在一个地方修改，比起修改代码的多个部分更简单高效。因此，使代码更加模块化和健壮。

此外，胖模型，瘦控制器也是 DRY 的体现，因为你把代码写在模型中，控制器只负责调用，比如：

```
# Post 模型
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?',
timestamp) }


# 任何控制器
def index
    ....
    @unpublished_posts = Post.unpublished
    ....
end

def others
    ...
    @unpublished_posts = Post.unpublished
    ...
end
```

这也有助于形成一种以API驱动的结构，其中内部方法被隐藏，变更通过以API方式传递参数来实现。

第6.6节：你不会需要它（YAGNI）

如果你能对某个功能说"YAGNI"（你不会需要它），那么你最好不要实现它。通过专注于简洁性，可以节省大量开发时间。无论如何实现这些功能可能会导致以下问题：

问题
过度设计

如果产品比必须的更复杂，那就是过度设计。通常这些"尚未使用"的功能永远不会以它们被编写的预期方式使用，如果它们被使用，必须进行重构。过早的优化，尤其是性能优化，往往导致未来被证明错误的设计决策。

代码膨胀

代码膨胀指不必要的复杂代码。例如，抽象、冗余或设计模式的错误应用都可能导致这种情况。代码库变得难以理解、混乱且维护成本高昂。

功能蔓延

- Many others

Section 6.5: Don't Repeat Yourself (DRY)

To help to maintain clean code, Rails follows the principle of DRY.

It involves whenever possible, re-using as much code as possible rather than duplicating similar code in multiple places (for example, using partials). This reduces *errors*, keeps your code *clean* and enforces the principle of *writing code once* and then reusing it. It is also easier and more efficient to update code in one place than to update multiple parts of the same code. Thus making your code more modular and robust.

Also *Fat Model, Skinny Controller* is DRY, because you write the code in your model and in the controller only do the call, like:

```
# Post model
scope :unpublished, ->(timestamp = Time.now) { where('published_at IS NULL OR published_at > ?',
timestamp) }


# Any controller
def index
    ....
    @unpublished_posts = Post.unpublished
    ....
end

def others
    ...
    @unpublished_posts = Post.unpublished
    ...
end
```

This also helps lead to an API driven structure where internal methods are hidden and changes are achieved through passing parameters in an API fashion.

Section 6.6: You Ain't Gonna Need it (YAGNI)

If you can say "YAGNI" (You ain't gonna need it) about a feature, you better not implement it. There can be a lot of development time saved through focussing onto simplicity. Implementing such features anyway can lead to problems:

Problems
Overengineering

If a product is more complicated than it has to be, it is over engineered. Usually these "not yet used" features will never be used in the intended way they were written and have to be refactored if they ever get used. Premature optimisations, especially performance optimisations, often lead to design decisions which will be proved wrong in the future.

Code Bloat

Code Bloat means unnecessary complicated code. This can occur for example by abstraction, redundancy or incorrect application of design patterns. The code base becomes difficult to understand, confusing and expensive to maintain.

Feature Creep

功能蔓延指添加超出产品核心功能的新特性，导致产品复杂度不必要地增加。

开发周期长

用于开发必要功能的时间被用来开发不必要的功能。产品交付时间更长。

解决方案

KISS——保持简单，愚蠢点没关系

根据KISS原则，大多数系统如果设计得简单，效果最好。简洁应作为主要设计目标以减少复杂性。例如，可以通过遵循“单一职责原则”来实现。

YAGNI——你不会需要它

少即是多。考虑每个功能，它真的有必要吗？如果你能想到任何理由说明它是YAGNI，就不要加进来。最好在真正需要时再开发它。

持续重构

产品在不断稳步改进。通过重构，我们可以确保产品按照最佳实践完成，不会退化成拼凑的作品。

Feature Creep refers to adding new features that go beyond the core functionality of the product and lead to an unnecessarily high complexity of the product.

Long development time

The time which could be used to develop necessary features is spent to develop unnecessary features. The product takes longer to deliver.

Solutions

KISS - Keep it simple, stupid

According to KISS, most systems work the best if they are designed simple. Simplicity should be a primary design goal to reduce complexity. It can be achieved by following the “Single Responsibility Principle” for example.

YAGNI – You Ain't Gonna Need it

Less is more. Think about every feature, is it really needed? If you can think of any way that it's YAGNI, leave it away. It's better to develop it when it's needed.

Continuous Refactoring

The product is being improved steadily. With refactoring, we can make sure that the product is being done according to best practice and does not degenerate to a patch work.

第七章：命名约定

第7.1节：控制器

控制器类名使用复数形式。原因是控制器控制多个对象实例。

例如：OrdersController 将是 orders 表的控制器。Rails 会在 /app/controllers 目录下查找名为 orders_controller.rb 的类定义文件。

例如：PostsController 将是 posts 表的控制器。

如果控制器类名包含多个大写单词，则假定表名在这些单词之间有下划线分隔。

*例如：*如果控制器名为 `PendingOrdersController`，则该控制器对应的文件名假定为 `pending_orders_controller.rb`。

第7.2节：模型

模型名称采用不间断的混合大小写类命名规范，且始终为表名的单数形式。

例如：如果表名为 `orders`，关联的模型名为 `Order`

例如：如果表名为 `posts`，关联的模型名为 `Post`

Rails 会在 /app/**models** 目录下查找名为 **order**.rb 的类定义文件。

如果模型类名包含多个大写单词，则假定表名在这些单词之间有下划线分隔。

例如：如果一个模型命名为BlogPost，则假定的表名将是blog_posts。

第7.3节：文件名和自动加载

Rails文件——以及Ruby文件通常——应使用lower_snake_case格式命名。例如

```
app/controllers/application_controller.rb
```

是包含ApplicationController类定义的文件。注意，虽然类和模块名称使用PascalCase，但它们所在的文件仍应使用lower_snake_case。

命名一致性很重要，因为Rails会根据需要自动加载文件，并使用"词形变化"在不同命名风格之间转换，例如将application_controller转换为
ApplicationController，反之亦然。

例如，如果Rails发现`BlogPost`类不存在（尚未加载），它会查找名为
`blog_post.rb`的文件并尝试加载该文件。

因此，给文件命名时也应与其内容相符，因为自动加载器期望文件名与内容匹配。例如，如果blog_post.rb中包含的类名仅为Post，则会出现LoadError错误：
期望[某路径]/blog_post.rb定义BlogPost。

如果你在app/something/下添加一个目录（例如 /models/products/），并且

Chapter 7: Naming Conventions

Section 7.1: Controllers

Controller class names are pluralized. The reason is the controller controls multiple instances of object instance.

For Example: `OrdersController` would be the controller for an `orders` table. Rails will then look for the class definition in a file called `orders_controller.rb` in the `/app/controllers` directory.

For Example: `PostsController` would be the controller for a `posts` table.

If the controller class name has multiple capitalized words, the table name is assumed to have underscores between these words.

For Example: If a controller is named `PendingOrdersController` then assumed file name for this controller will be `pending_orders_controller.rb`.

Section 7.2: Models

The model is named using the class naming convention of unbroken MixedCase and is always the singular of the table name.

For Example: If a table was named `orders`, the associated model would be named `Order`

For Example: If a table was named `posts`, the associated model would be named `Post`

Rails will then look for the class definition in a file called `order.rb` in the `/app/models` directory.

If the model class name has multiple capitalized words, the table name is assumed to have underscores between these words.

For Example: If a model is named `BlogPost` then assumed table name will be `blog_posts`.

Section 7.3: Filenames and autoloading

Rails files - and Ruby files in general - should be named with `lower_snake_case` filenames. E.g.

```
app/controllers/application_controller.rb
```

is the file that contains the `ApplicationController` class definition. Note that while `PascalCase` is used for class and module names, the files in which they reside should still be `lower_snake_case`.

Consistent naming is important since Rails makes use of auto-loading files as needed, and uses "inflection" to transform between different naming styles, such as transforming `application_controller` to `ApplicationController` and back again.

E.g. if Rails sees that the `BlogPost` class doesn't exist (hasn't been loaded yet), it'll look for a file named `blog_post.rb` and attempt to load that file.

It is therefore also important to name files for what they contain, since the autoloader expects file names to match content. If, for instance, the `blog_post.rb` instead contains a class named just `Post`, you'll see a **`LoadError`**: `Expected [some path]/blog_post.rb to define BlogPost`.

If you add a dir under `app/something/` (e.g. /models/products/), and

- 如果想要在新目录中为模块和类命名空间，则无需做任何操作，它会自动加载。例如，在app/models/products/目录下，你需要将你的class包裹在moduleProducts`中。
- 如果不想在新目录中为模块和类命名空间，则必须添加 config.autoload_paths += %W(#{config.root}/app/models/products) 到你的application.rb以实现自动加载。

还有一点需要注意（尤其是当英语不是你的第一语言时），Rails 会考虑英语中不规则复数名词的情况。所以如果你有一个名为“Foot”的模型，相关的控制器需要命名为“FeetController”，而不是“FootsController”，这样Rails的“魔法”路由（以及许多其他功能）才能正常工作。

第7.4节：视图和布局

当控制器动作被渲染时，Rails 会尝试根据控制器的名称找到匹配的布局和视图。

视图和布局放置在app/views目录下。

对于对PeopleController#index动作的请求，Rails 会搜索：

- 在app/views/layouts/目录下名为**people**的布局（如果未找到匹配，则使用application布局）
- 默认在app/views/people/**目录下名为**index.**htmlerb**的视图
- 如果你想渲染名为index_new.html.erb的其他文件，则必须在PeopleController#index动作中写代码，如 render 'index_new'
- 我们可以通过编写 render 'index_new', layout: 来为每个 action 设置不同的 layouts 'your_layout_name'

第7.5节：从控制器名称获取模型类

你可以通过以下方式从控制器名称获取模型类（上下文为控制器类）：

```
class MyModelController < ActionController::Base

  # 返回此控制器对应的模型类
  # @return [ActiveRecord::Base]
  def corresponding_model_class
    # ... 添加一些验证
    controller_name.classify.constantize
  end
end
```


- want to namespace modules and classes inside new dir then you don't need to do anything and it'll be loaded itself. For example, in app/models/products/ you would need to wrap your class inmodule Products`.
- don't want to namespace modules and classes inside my new dir then you have to add config.autoload_paths += %W(#{config.root}/app/models/products) to your application.rb to autoload.

One more thing to pay attention to (especially if English is not your first language) is the fact that Rails accounts for irregular plural nouns in English. So if you have model named "Foot" the corresponding controller needs to be called "FeetController" rather than "FootsController" if you want rails "magic" routing (and many more such features) to work.

Section 7.4: Views and Layouts

When a controller action is rendered, Rails will attempt to find a matching layout and view based on the name of the controller.

Views and layouts are placed in the app/views directory.

Given a request to the PeopleController#index action, Rails will search for:

- the layout called people in app/views/layouts/ (or application if no match is found)
- a view called index.html.erb in app/views/people/ by default
- if you wish to render other file called index_new.html.erb you have to write code for that in PeopleController#index action like render 'index_new'
- we can set different layouts for every action by writing render 'index_new', layout: 'your_layout_name'

Section 7.5: Models class from Controller name

You can get a Model class from a Controller name this way (context is Controller class):

```
class MyModelController < ActionController::Base

  # Returns corresponding model class for this controller
  # @return [ActiveRecord::Base]
  def corresponding_model_class
    # ... add some validation
    controller_name.classify.constantize
  end
end
```

第8章：ActionCable

第8.1节：用户认证

```
# app/channels/application_cable/connection.rb
模块 ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      self.current_user = find_verified_user
      logger.add_tags 'ActionCable', current_user.id
      # 可以用用户名、ID、邮箱等替换 current_user.id
    end

    protected

    def find_verified_user
      if verified_user = env['warden'].user
       verified_user
      else
        reject_unauthorized_connection
      结束
    结束
  end
end
```

第8.2节：[基础] 服务器端

```
# app/channels/appearance_channel.rb
class NotificationsChannel < ApplicationCable::Channel
  def subscribed
从 "notifications" 订阅流
  end

  def 取消订阅
  end

  def 通知(data)
ActionCable.server.broadcast "notifications", { title: '新消息！', body: data }
  结束
结束
```

第8.3节：[基础] 客户端（CoffeeScript）

app/assets/javascripts/channels/notifications.coffee

```
App.notifications = App.cable.subscriptions.create "NotificationsChannel",
  connected: ->
    # 当服务器上的订阅准备好使用时调用
$(document).on "change", "input", (e)=>
      @notify(e.target.value)

断开连接: ->
    # 当服务器终止订阅时调用
    $(document).off "change", "input"

  接收到：(data) ->
    # 当该频道的 websocket 有传入数据时调用
```


Chapter 8: ActionCable

Section 8.1: User Authentication

```
# app/channels/application_cable/connection.rb
module ApplicationCable
  class Connection < ActionCable::Connection::Base
    identified_by :current_user

    def connect
      self.current_user = find_verified_user
      logger.add_tags 'ActionCable', current_user.id
      # Can replace current_user.id with usernames, ids, emails etc.
    end

    protected

    def find_verified_user
      if verified_user = env['warden'].user
        verified_user
      else
        reject_unauthorized_connection
      end
    end
  end
end
```

Section 8.2: [Basic] Server Side

```
# app/channels/appearance_channel.rb
class NotificationsChannel < ApplicationCable::Channel
  def subscribed
    stream_from "notifications"
  end

  def unsubscribed
  end

  def notify(data)
    ActionCable.server.broadcast "notifications", { title: 'New things!', body: data }
  end
end
```

Section 8.3: [Basic] Client Side (Coffeescript)

app/assets/javascripts/channels/notifications.coffee

```
App.notifications = App.cable.subscriptions.create "NotificationsChannel",
  connected: ->
    # Called when the subscription is ready for use on the server
    $(document).on "change", "input", (e)=>
      @notify(e.target.value)

  disconnected: ->
    # Called when the subscription has been terminated by the server
     $(document).off "change", "input"

  received: (data) ->
    # Called when there's incoming data on the websocket for this channel
```

```
$('body').append(data)

  通知：(data)->
    @perform('notify', data: data)
```

app/assets/javascripts/application.js # 通常是这样生成的

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

app/assets/javascripts/cable.js # 通常是这样生成的

```
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```



```
    $('body').append(data)

  notify: (data)->
    @perform('notify', data: data)
```

app/assets/javascripts/application.js # usually generated like this

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

app/assets/javascripts/cable.js # usually generated like this

```
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
  this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```

第9章：ActiveModel

第9.1节：使用 ActiveModel::Validations

你可以验证任何对象，甚至是普通的 Ruby 对象。

```
类 User
  包含 ActiveModel::Validations

  attr_reader :name, :age

  定义 initialize(name, age)
    @name = name
    @age  = age
  end

验证 :name, presence: true
验证 :age, numericality: { only_integer: true, greater_than: 12 }
end
```

```
用户。new('约翰·史密斯', 28).有效? #=> true
用户。new('简·史密斯', 11).有效? #=> false
用户。new(nil, 30).有效?          #=> false
```


Chapter 9: ActiveModel

Section 9.1: Using ActiveModel::Validations

You can validate any object, even plain ruby.

```
class User
  include ActiveModel::Validations

  attr_reader :name, :age

  def initialize(name, age)
    @name = name
    @age  = age
  end

  validates :name, presence: true
  validates :age, numericality: { only_integer: true, greater_than: 12 }
end
```

```
User.new('John Smith', 28).valid? #=> true
User.new('Jane Smith', 11).valid? #=> false
User.new(nil, 30).valid?          #=> false
```

第10章：Rails中的用户认证

Devise是一个非常强大的gem，安装后即可提供注册、登录和登出功能。此外，用户可以为其应用添加认证和限制。Devise还自带视图，用户如果愿意可以使用。用户也可以根据自己的需求和要求自定义注册和登录表单。需要注意的是，Devise建议如果你是Rails新手，最好自己实现登录功能。

第10.1节：使用Devise进行认证

将gem添加到Gemfile中：

```
gem 'devise'
```

然后运行bundle install命令。

使用命令$ rails generate devise:install生成所需的配置文件。

在每个环境中为Devise邮件发送器设置默认URL选项。在开发环境中添加以下行：

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

到你的config/environments/development.rb

同样，在生产环境中编辑config/environments/production.rb文件并添加

```
config.action_mailer.default_url_options = { host: 'your-site-url'}
```

然后使用以下命令创建模型：$ rails generate devise USER 其中 USER 是你想要实现认证的类名。

最后，运行：rake db:migrate 即可完成设置。

自定义视图

如果你需要配置视图，可以使用 rails generate devise:views 生成器，它会将所有视图复制到你的应用中。然后你可以根据需要编辑它们。

如果你的应用中有多个 Devise 模型（例如 User 和 Admin），你会发现 Devise 对所有模型使用相同的视图。Devise 提供了一种简单的方法来自定义视图。在 config/initializers/devise.rb 文件中设置 config.scoped_views 为 true。 = true 在 config/initializers/devise.rb 文件中。

你也可以使用生成器来创建作用域视图：rails generate devise:views users如果你只想生成部分视图，比如

registerable 和 confirmable 模块的视图，可以使用 -v 参数：rails generate devise:views -v registrations confirmations

第10.2节：Devise 控制器过滤器与辅助方法

要使用 devise 设置带用户认证的控制器，添加以下 before_action：（假设你的 devise 模型是 'User'）：

```
before_action :authenticate_user!
```


Chapter 10: User Authentication in Rails

Devise is a very powerful gem, it allows you to sign up, sign in and sign out options just after installing. Moreover user can add authentications and restrictions to its applications. Devise also come with its own views, if user wants to use. A user can also customize sign up and sign in forms according to its need and requirement. It should be noted that Devise recommends that you implement your own login if you're new to rails.

Section 10.1: Authentication using Devise

Add gem to the Gemfile:

```
gem 'devise'
```

Then run the `bundle install` command.

Use command `$ rails generate devise:install` to generate required configuration file.

Set up the default URL options for the Devise mailer in each environment In development environment add this line:

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

to your config/environments/development.rb

similarly in production this edit config/environments/production.rb file and add

```
config.action_mailer.default_url_options = { host: 'your-site-url'}
```

Then create a model using:`$ rails generate devise USER` Where USER is the class name for which you want to implement authentication.

Finally, run: `rake db:migrate` and you are all set.

Custom views

If you need to configure your views, you can use the `rails generate devise:views` generator that will copy all views to your application. Then you can edit them as desired.

If you have more than one Devise model in your application (for example User and Admin), you will notice that Devise uses the same views for all models. Devise offers an easy way to customize views. Set `config.scoped_views = true` inside the config/initializers/devise.rb file.

You can also use the generator to create scoped views: `rails generate devise:views users`

If you would like to generate only a few sets of views, such as the ones for the registerable and confirmable module use the -v flag: `rails generate devise:views -v registrations confirmations`

Section 10.2: Devise Controller Filters & Helpers

To set up a controller with user authentication using devise, add this before_action: (assuming your devise model is 'User'):

```
before_action :authenticate_user!
```

要验证用户是否已登录，请使用以下辅助方法：

`user_signed_in?`

对于当前已登录的用户，请使用此辅助方法：

`current_user`

您可以访问此作用域的会话：

`user_session`

- 请注意，如果您的 Devise 模型名为Member而不是User，则将上述的user替换为member

第10.3节：Omniauth

首先选择您的认证策略并将其添加到您的Gemfile中。您可以在这里找到策略列表：
https://github.com/intridea/omniauth/wiki/List-of-Strategies

```
gem 'omniauth-github', :github => 'intridea/omniauth-github'
gem 'omniauth-openid', :github => 'intridea/omniauth-openid'
```

你可以这样将其添加到你的 Rails 中间件：

```
Rails.application.config.middleware.use OmniAuth::Builder do
  require 'openid/store/filesystem'
provider :github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
  provider :openid, :store => OpenID::Store::Filesystem.new('/tmp')
end
```

默认情况下，OmniAuth 会将 /auth/:provider 添加到你的路由中，你可以开始使用这些路径。

默认情况下，如果发生失败，omniauth 会重定向到 /auth/failure

第10.4节：has_secure_password

创建用户模型

```
rails generate model User email:string password_digest:string
```

向 User 模型添加 has_secure_password 模块
class User < ActiveRecord::Base has_secure_password end

现在你可以创建带密码的新用户

```
user = User.new email: 'bob@bob.com', password: 'Password1', password_confirmation: 'Password1'
```

使用 authenticate 方法验证密码

```
user.authenticate('somepassword')
```

第10.5节：has_secure_token

创建用户模型

```
# 模式：User(token:string, auth_token:string)
```


To verify if a user is signed in, use the following helper:

`user_signed_in?`

For the current signed-in user, use this helper:

`current_user`

You can access the session for this scope:

`user_session`

- Note that if your Devise model is called `Member` instead of `User`, replace `user` above with `member`

Section 10.3: Omniauth

First choose your auth strategy and add it to your `Gemfile`. You can find a list of strategies here:
https://github.com/intridea/omniauth/wiki/List-of-Strategies

```
gem 'omniauth-github', :github => 'intridea/omniauth-github'
gem 'omniauth-openid', :github => 'intridea/omniauth-openid'
```

You can add this to your rails middleware like so:

```
Rails.application.config.middleware.use OmniAuth::Builder do
  require 'openid/store/filesystem'
  provider :github, ENV['GITHUB_KEY'], ENV['GITHUB_SECRET']
  provider :openid, :store => OpenID::Store::Filesystem.new('/tmp')
end
```

By default, OmniAuth will add `/auth/:provider` to your routes and you can start by using these paths.

By default, if there is a failure, omniauth will redirect to `/auth/failure`

Section 10.4: has_secure_password

Create User Model

```
rails generate model User email:string password_digest:string
```

Add has_secure_password module to User model
class User < ActiveRecord::Base has_secure_password end

Now you can create a new user with password

```
user = User.new email: 'bob@bob.com', password: 'Password1', password_confirmation: 'Password1'
```

Verify password with authenticate method

```
user.authenticate('somepassword')
```

Section 10.5: has_secure_token

Create User Model

```
# Schema: User(token:string, auth_token:string)
```

```ruby
class User < ActiveRecord::Base
  has_secure_token
has_secure_token :auth_token
end
```

现在，当你创建一个新用户时，token 和 auth_token 会自动生成

```ruby
user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
user.auth_token # => "77TMHrHJFvFDwodq8w7Ev2m7"
```

您可以使用regenerate_token和regenerate_auth_token来更新令牌

```ruby
user.regenerate_token # => true
user.regenerate_auth_token # => true
```



```ruby
class User < ActiveRecord::Base
  has_secure_token
  has_secure_token :auth_token
end
```

Now when you create a new user a token and auth_token are automatically generated

```ruby
user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
user.auth_token # => "77TMHrHJFvFDwodq8w7Ev2m7"
```

You can update the tokens using `regenerate_token` and `regenerate_auth_token`

```ruby
user.regenerate_token # => true
user.regenerate_auth_token # => true
```

第11章：ActiveRecord关联

第11.1节：多态关联

这种类型的关联允许ActiveRecord模型属于多种模型记录。常见
示例：

```
class Human < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Company < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Address < ActiveRecord::Base
  belongs_to :addressable, :polymorphic => true
end
```

如果没有这种关联，您的Address表中会有所有这些外键，但您实际上只会为其中一个外键有值，因为在这种情况下
，一个地址只能属于一个实体（Human或Company）。
它看起来会是这样的：

```
class Address < ActiveRecord::Base
  belongs_to :human
belongs_to :company
end
```

第11.2节：自引用关联

自引用关联用于将模型与其自身关联。最常见的例子是管理朋友与其关注者之间的关联。

示例。

```
rails g model friendship user_id:references friend_id:integer
```

现在你可以这样关联模型；

```
class User < ActiveRecord::Base
  has_many :friendships
has_many :friends, :through => :friendships
  has_many :inverse_friendships, :class_name => "Friendship", :foreign_key => "friend_id"
  has_many :inverse_friends, :through => :inverse_friendships, :source => :user
end
```

另一个模型将如下所示；

```
class Friendship < ActiveRecord::Base
  belongs_to :user
belongs_to :friend, :class_name => "User"
end
```


Chapter 11: ActiveRecord Associations

Section 11.1: Polymorphic association

This type of association allows an ActiveRecord model to belong to more than one kind of model record. Common example:

```
class Human < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Company < ActiveRecord::Base
  has_one :address, :as => :addressable
end

class Address < ActiveRecord::Base
  belongs_to :addressable, :polymorphic => true
end
```

Without this association, you'd have all these foreign keys in your Address table but you only would ever have a value for one of them because an address, in this scenario, can only belong to one entity (Human or Company). Here is what it would look like:

```
class Address < ActiveRecord::Base
  belongs_to :human
  belongs_to :company
end
```

Section 11.2: Self-Referential Association

Self-referential association is used to associate a model with itself. The most frequent example would be, to manage association between a friend and his follower.

ex.

```
rails g model friendship user_id:references friend_id:integer
```

now you can associate models like;

```
class User < ActiveRecord::Base
  has_many :friendships
  has_many :friends, :through => :friendships
  has_many :inverse_friendships, :class_name => "Friendship", :foreign_key => "friend_id"
  has_many :inverse_friends, :through => :inverse_friendships, :source => :user
end
```

and the other model will look like;

```
class Friendship < ActiveRecord::Base
  belongs_to :user
  belongs_to :friend, :class_name => "User"
end
```

第11.3节：belongs_to

belongs_to关联建立了与另一个模型的一对一连接，因此声明模型的每个实例“属于”另一个模型的一个实例。

例如，如果你的应用包含用户和帖子，并且每个帖子只能分配给一个用户，你可以这样声明帖子模型：

```
class Post < ApplicationRecord
  belongs_to :user
end
```

在你的表结构中，你可能会有

```
create_table "posts", force: :cascade do |t|
t.integer  "user_id",  limit: 4
end
```

第11.4节：has_one

has_one关联建立了与另一个模型的一对一连接，但语义不同。该关联表示模型的每个实例包含或拥有另一个模型的一个实例。

例如，如果你的应用中每个用户只有一个账户，你可以这样声明用户模型：

```
class User < ApplicationRecord
  has_one :account
end
```

在 Active Record 中，当你有一个 has_one 关联时，Active Record 确保只有一条记录存在该外键。

在我们的示例中：在 accounts 表中，对于特定的 user_id 只能有一条记录。如果你尝试为同一个用户关联更多账户，它会将之前条目的外键设为 null（使其成为孤立记录），并自动创建一个新的账户。即使新条目的保存失败，它也会将之前的条目标记为 null，以保持数据一致性。

```
user = User.first
user.build_account(name: "sample")
user.save   [成功保存，并在accounts 表中创建一条 user_id 为 1 的记录]
user.build_account(name: "sample1") [自动将之前条目的外键设为 null]
user.save [创建一个名称为 sample1 且 user_id 为 1 的新账户]
```

第11.5节：has_many

一个 has_many 关联表示与另一个模型的一对多连接。该关联通常位于 belongs_to 关联的另一端。

该关联表示模型的每个实例可以拥有零个或多个另一个模型的实例。

例如，在包含用户和帖子（posts）的应用中，用户模型可以这样声明：

```
class User < ApplicationRecord
  has_many :posts
```


Section 11.3: belongs_to

A `belongs_to` association sets up a one-to-one connection with another model, so each instance of the declaring model "belongs to" one instance of the other model.

For example, if your application includes users and posts, and each post can be assigned to exactly one user, you'd declare the post model this way:

```
class Post < ApplicationRecord
  belongs_to :user
end
```

In your table structure you might then have

```
create_table "posts", force: :cascade do |t|
  t.integer  "user_id",  limit: 4
end
```

Section 11.4: has_one

A `has_one` association sets up a one-to-one connection with another model, but with different semantics. This association indicates that each instance of a model contains or possesses one instance of another model.

For example, if each user in your application has only one account, you'd declare the user model like this:

```
class User < ApplicationRecord
  has_one :account
end
```

In Active Record, when you have a `has_one` relation, active record ensures that the only one record exists with the foreign key.

Here in our example: In accounts table, there can only be one record with a particular user_id. If you try to associate one more account for the same user, it makes the previous entry's foreign key as null(making it orphan) and creates a new one automatically. It makes the previous entry null even if the save fails for the new entry to maintain consistency.

```
user = User.first
user.build_account(name: "sample")
user.save   [Saves it successfully, and creates an entry in accounts table with user_id 1]
user.build_account(name: "sample1")  [automatically makes the previous entry's foreign key null]
user.save  [creates the new account with name sample 1 and user_id 1]
```

Section 11.5: has_many

A `has_many` association indicates a one-to-many connection with another model. This association generally is located 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 users and posts, the user model could be declared like this:

```
class User < ApplicationRecord
  has_many :posts
```

```
end
```

Post 表的结构将保持与 belongs_to 示例中相同；相比之下，User 不需要任何模式更改。

如果你想获取该用户发布的所有帖子列表，那么你可以添加以下内容（即你可以为关联对象添加作用域）：

```
class 用户 < ApplicationRecord
has_many :published_posts, -> { where("posts.published IS TRUE") }, class_name: "Post"
end
```

第11.6节：has_many :through 关联

has_many :through 关联通常用于与另一个模型建立多对多关系。该关联表示声明模型可以通过第三个模型与另一个模型的零个或多个实例匹配。

例如，考虑一个医疗机构，患者预约看医生。相关的关联声明可能如下所示：

```
class 医生 < ApplicationRecord
  has_many :appointments
has_many :patients, through: :appointments
end

class 预约 < ApplicationRecord
  belongs_to :physician
belongs_to :patient
end

class Patient < ApplicationRecord
  has_many :appointments
has_many :physicians, 通过: :appointments
end
```

第11.7节：has_one :through 关联

has_one :through 关联建立了与另一个模型的一对一连接。该关联表示声明模型可以通过第三个模型与另一个模型的一个实例匹配。

例如，如果每个 supplier 有一个 account，且每个账户都关联一个账户历史，那么 supplier 模型可以如下所示：

```
class Supplier < ApplicationRecord
  has_one :account
has_one :account_history, 通过: :account
end

class Account < ApplicationRecord
  belongs_to :supplier
has_one :account_history
end

class AccountHistory < ApplicationRecord
  belongs_to :account
```



```
end
```

The table structure of Post would remain the same as in the belongs_to example; in contrast, User would not require any schema changes.

If you want to get the list of all the published posts for the User, then you can add the following (i.e. you can add scopes to your association objects):

```
class User < ApplicationRecord
  has_many :published_posts, -> { where("posts.published IS TRUE") }, class_name: "Post"
end
```

Section 11.6: 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 < ApplicationRecord
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ApplicationRecord
  belongs_to :physician
  belongs_to :patient
end

class Patient < ApplicationRecord
  has_many :appointments
  has_many :physicians, through: :appointments
end
```

Section 11.7: 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 supplier model could look like this:

```
class Supplier < ApplicationRecord
  has_one :account
  has_one :account_history, through: :account
end

class Account < ApplicationRecord
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ApplicationRecord
  belongs_to :account
```

```
end
```

第11.8节：has_and_belongs_to_many 关联

A has_and_belongs_to_many 关联创建了与另一个模型的直接多对多连接，中间没有中介模型。

例如，如果您的应用包含装配件和零件，每个装配件有多个零件，每个零件出现在多个装配件中，您可以这样声明模型：

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```



```
end
```

Section 11.8: 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 < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

第12章：ActiveRecord 验证

第12.1节：验证属性长度

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

可能的长度约束选项有：

- :minimum - 属性长度不能小于指定值。
- :maximum - 属性长度不能大于指定值。
- :in（或:within）- 属性长度必须包含在给定区间内。该选项的值必须是一个范围。

- :is - 属性长度必须等于给定的值。

第12.2节：验证属性的格式

使用format和with选项验证属性值是否匹配正则表达式。

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

你也可以定义一个常量，将其值设置为正则表达式，并传递给with:选项。这对于非常复杂的正则表达式可能更方便

```
PHONE_REGEX = /\A\(\d{3}\)\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

默认的错误信息是is invalid。可以通过:message选项更改此信息。

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

反向也适用，你可以使用without:指定某个值不应匹配某个正则表达式：

option

第12.3节：验证属性的存在性

此辅助方法验证指定的属性不为空。

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

你可以使用absence辅助方法来验证指定属性的缺失。它使用present?方法来检查nil或空值。

Chapter 12: ActiveRecord Validations

Section 12.1: Validating length of an attribute

```
class Person < ApplicationRecord
  validates :name, length: { minimum: 2 }
  validates :bio, length: { maximum: 500 }
  validates :password, length: { in: 6..20 }
  validates :registration_number, length: { is: 6 }
end
```

The possible length constraint options are:

- `:minimum` - The attribute cannot have less than the specified length.
- `:maximum` - The attribute cannot have more than the specified length.
- `:in` (or `:within`) - The attribute length must be included in a given interval. The value for this option must be a range.
- `:is` - The attribute length must be equal to the given value.

Section 12.2: Validates format of an attribute

Validate that an attribute's value matches a regular expression using `format` and the `with` option.

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

You can also define a constant and set its value to a regular expression and pass it to the `with:` option. This might be more convenient for really complex regular expressions

```
PHONE_REGEX = /\A\(\d{3}\)\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

The default error message is `is invalid`. This can be changed with the `:message` option.

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

The reverse also replies, and you can specify that a value should *not* match a regular expression with the `without:` option

Section 12.3: Validating presence of an attribute

This helper validates that the specified attributes are not empty.

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

You can use the `absence` helper to validate that the specified attributes are absent. It uses the `present?` method to check for nil or empty values.

```ruby
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

注意：如果属性是一个boolean类型，不能使用常规的存在性验证（属性为false时验证不通过）。你可以通过使用包含验证来实现：

```ruby
validates :attribute, inclusion: [true, false]
```

第12.4节：自定义验证

你可以通过添加继承自`ActiveModel::Validator`或`ActiveModel::EachValidator`的新类来添加自定义验证。这两种方法类似，但工作方式略有不同：

ActiveModel::Validator 和 validates_with

实现`validate`方法，该方法以一个记录（record）作为参数并对其进行验证。然后在模型上使用`validates_with`和该类。

```ruby
# app/validators/starts_with_a_validator.rb
class StartsWithAValidator < ActiveModel::Validator
  def validate(record)
    除非 record.name.starts_with? 'A'
      record.errors[:name] << '名字必须以A开头！'
    end
  end
end

class Person < ApplicationRecord
  validates_with StartsWithAValidator
end
```

ActiveModel::EachValidator 和 validate

如果你更喜欢使用常见的validate方法对单个参数进行验证，可以创建一个继承自ActiveModel::EachValidator的类，并实现validate_each方法，该方法接受三个参数：record、attribute和value：

```ruby
class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    除非 value =~ /\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
      record.errors[attribute] << (options[:message] || '不是有效的邮箱地址')
    end
  end
end

class Person < ApplicationRecord
  validates :email, presence: true, email: true
end
```

更多信息请参见Rails guides。

第12.5节：验证属性的包含性

您可以使用`inclusion:`辅助方法来检查一个值是否包含在数组中。选项：`in`及其别名`:within`显示可接受值的集合。


```ruby
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

Note: In case the attribute is a `boolean` one, you cannot make use of the usual presence validation (the attribute would not be valid for a `false` value). You can get this done by using an inclusion validation:

```ruby
validates :attribute, inclusion: [true, false]
```

Section 12.4: Custom validations

You can add your own validations adding new classes inheriting from `ActiveModel::Validator` or from `ActiveModel::EachValidator`. Both methods are similar but they work in a slightly different ways:

ActiveModel::Validator and validates_with

Implement the `validate` method which takes a record as an argument and performs the validation on it. Then use `validates_with` with the class on the model.

```ruby
# app/validators/starts_with_a_validator.rb
class StartsWithAValidator < ActiveModel::Validator
  def validate(record)
    unless record.name.starts_with? 'A'
      record.errors[:name] << 'Need a name starting with A please!'
    end
  end
end

class Person < ApplicationRecord
  validates_with StartsWithAValidator
end
```

ActiveModel::EachValidator and validate

If you prefer to use your new validator using the common `validate` method on a single param, create a class inheriting from `ActiveModel::EachValidator` and implement the `validate_each` method which takes three arguments: `record`, `attribute`, and `value`:

```ruby
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 < ApplicationRecord
  validates :email, presence: true, email: true
end
```

More information on the Rails guides.

Section 12.5: Validates inclusion of an attribute

You can check if a value is included in an array using the `inclusion:` helper. The `:in` option and its alias, `:within` show the set of acceptable values.

```
class Country < ApplicationRecord
validates :continent, inclusion: { in: %w(非洲 南极洲 亚洲 澳大利亚
                                    欧洲 北美洲 南美洲) }
end
```

要检查一个值是否不包含在数组中，请使用exclusion:辅助方法

```
class 用户 < ApplicationRecord
validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

第12.6节：分组验证

有时让多个验证使用同一个条件是很有用的。可以通过with_options轻松实现。

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  结束
结束
```

只要满足条件:is_admin?，with_options块内的所有验证都会自动通过该条件

第12.7节：验证属性的数值性

此验证限制只能插入数值。

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

除了 :only_integer 之外，该辅助方法还接受以下选项以添加对可接受值的约束：

- :greater_than - 指定值必须大于提供的值。该选项的默认错误信息为 "must be greater than %{count}"（必须大于 %{count}）。
- :greater_than_or_equal_to - 指定值必须大于或等于提供的值。该选项的默认错误信息为 "must be greater than or equal to %{count}"（必须大于或等于 %{count}）。
- :equal_to - 指定值必须等于提供的值。该选项的默认错误信息为 "must be equal to %{count}"（必须等于 %{count}）。
- :less_than - 指定值必须小于提供的值。该选项的默认错误信息为 "must be less than %{count}"（必须小于 %{count}）。
- :less_than_or_equal_to - 指定值必须小于或等于提供的值。该选项的默认错误信息为 "must be less than or equal to %{count}"（必须小于或等于 %{count}）。
- :other_than - 指定值必须不等于提供的值。该选项的默认错误信息为 "must be other than %{count}"（必须不等于 %{count}）。
- :odd - 如果设置为 true，指定值必须是奇数。该选项的默认错误信息为 "must be odd"（必须是奇数）。

- :even - 指定如果设置为 true，则值必须是偶数。该选项的默认错误信息是“必须是偶数”。

> 默认情况下，numericality 不允许 nil 值。你可以使用 allow_nil: true 选项来允许它。


```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                    Europe North America South America) }
end
```

To check if a value is not included in an array, use the `exclusion:` helper

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

Section 12.6: Grouping validation

Sometimes it is useful to have multiple validations use one condition. It can be easily achieved using with_options.

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  end
end
```

All validations inside of the with_options block will have automatically passed the condition if: :is_admin?

Section 12.7: Validating numericality of an attribute

This validation restricts the insertion of only numeric values.

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

Besides **`:only_integer`**, this helper also accepts the following options to add constraints to acceptable values:

- **`:greater_than`** - Specifies the value must be greater than the supplied value. The default error message for this option is "must be greater than %{count}".
- **`:greater_than_or_equal_to`** - Specifies the value must be greater than or equal to the supplied value. The default error message for this option is "must be greater than or equal to %{count}".
- **`:equal_to`** - Specifies the value must be equal to the supplied value. The default error message for this option is "must be equal to %{count}".
- **`:less_than`** - Specifies the value must be less than the supplied value. The default error message for this option is "must be less than %{count}".
- **`:less_than_or_equal_to`** - Specifies the value must be less than or equal to the supplied value. The default error message for this option is "must be less than or equal to %{count}".
- **`:other_than`** - Specifies the value must be other than the supplied value. The default error message for this option is "must be other than %{count}".
- **`:odd`** - Specifies the value must be an odd number if set to true. The default error message for this option is "must be odd".
- **`:even`** - Specifies the value must be an even number if set to true. The default error message for this option is "must be even".

> By default, numericality doesn't allow nil values. You can use allow_nil: true option to permit it.

第12.8节：验证属性的唯一性

此辅助方法验证属性的值在对象保存之前是唯一的。

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

有一个 :scope 选项，你可以用它来指定一个或多个属性，用于限制唯一性检查：

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "每年只能发生一次" }
end
```

还有一个 :case_sensitive 选项，你可以用它来定义唯一性约束是否区分大小写。该选项默认为 true。

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

第12.9节：跳过验证

如果您想跳过验证，请使用以下方法。这些方法即使对象无效，也会将其保存到数据库中。

- 递减！
- 递减计数器
- 递增！
- 递增计数器
- 切换！
- 触摸
- 更新全部
- 更新属性
- 更新列
- 更新多列
- 更新计数器

你也可以通过传递validate作为参数给save来跳过验证

```
User.save(validate: false)
```

第12.10节：属性确认

当你有两个文本字段需要接收完全相同的内容时，应使用此方法。例如，你可能想确认电子邮件地址或密码。此验证会创建一个***虚拟***属性，其名称是需要确认的字段名后面加上_confirmation。

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```


Section 12.8: Validate uniqueness of an attribute

This helper validates that the attribute's value is unique right before the object gets saved.

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

There is a `:scope` option that you can use to specify one or more attributes that are used to limit the uniqueness check:

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

There is also a `:case_sensitive` option that you can use to define whether the uniqueness constraint will be case sensitive or not. This option defaults to `true`.

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

Section 12.9: Skipping Validations

Use following methods if you want to skip the validations. These methods will save the object to the database even if it is invalid.

- decrement!
- decrement_counter
- increment!
- increment_counter
- toggle!
- touch
- update_all
- update_attribute
- update_column
- update_columns
- update_counters

You can also skip validation while saving by passing `validate` as an argument to `save`

```
User.save(validate: false)
```

Section 12.10: Confirmation of attribute

You should use this when you have two text fields that should receive exactly the same content. For example, you may want to confirm an email address or a password. This validation creates a ***virtual*** attribute whose name is the name of the field that has to be confirmed with `_confirmation` appended.

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

注意 仅当 email_confirmation 不为 nil 时才执行此检查。

要要求确认，请确保为确认属性添加存在性检查。

```
class Person < ApplicationRecord
  validates :email,      confirmation: true
  validates :email_confirmation, presence: true
end
```

来源

第12.11节：使用 :on 选项

:on 选项允许你指定验证应在何时进行。所有内置验证辅助方法的默认行为是在保存时运行（无论是创建新记录还是更新记录）。

```
class Person < ApplicationRecord
  # 允许使用重复的值更新邮箱
  validates :email, uniqueness: true, on: :create

  # 允许创建记录时年龄不是数字
validates :age, numericality: true, on: :update

  # 默认（在创建和更新时都验证）
验证 :name 的存在性，必须为 true
结束
```

第12.12节：条件验证

有时您可能只需要在某些条件下验证记录。

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
这里是返回布尔值的条件
  结束
结束
```

如果您的条件非常简单，可以使用 Proc：

```
class 用户 < ApplicationRecord
validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

对于否定条件，您可以使用 unless：

```
class 用户 < ApplicationRecord
validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

您也可以传递一个字符串，该字符串将通过 instance_eval 执行：

```
class 用户 < ApplicationRecord
验证:first_name的存在性，条件是'last_name.blank?'
end
```


Note This check is performed only if `email_confirmation` is not nil.

To require confirmation, make sure to add a presence check for the confirmation attribute.

```
class Person < ApplicationRecord
  validates :email,      confirmation: true
  validates :email_confirmation, presence: true
end
```

Source

Section 12.11: Using :on option

The `:on` option lets you specify when the validation should happen. The default behavior for all the built-in validation helpers is to be run on save (both when you're creating a new record and when you're updating it).

```
class Person < ApplicationRecord
  # 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
end
```

Section 12.12: Conditional validation

Sometimes you may need to validate record only under certain conditions.

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

If you conditional is really small, you can use a Proc:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

For negative conditional you can use `unless`:

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

You can also pass a string, which will be executed via `instance_eval`:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

第13章：ActiveRecord查询接口

ActiveRecord是MVC中的M，负责表示业务数据和逻辑的系统层。
将应用程序中的丰富对象连接到关系数据库管理系统中的表的技术是对象关系映射器（ORM）。

ActiveRecord会为你执行数据库查询，并且兼容大多数数据库系统。
无论你使用哪种数据库系统，ActiveRecord的方法格式始终相同。

第13.1节：.where

where方法可用于任何ActiveRecord模型，允许查询数据库中符合给定条件的一组记录。

where方法接受一个哈希，键对应模型所代表表的列名。

作为一个简单示例，我们将使用以下模型：

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

查找所有名字为Sven的人：

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

查找所有名字为Sven且姓氏为Schrodinger的人：

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

在上述示例中，sql输出显示只有当first_name和
last_name都匹配时，才会返回记录。

带有OR条件的查询

查找first_name == 'Bruce' 或 last_name == 'Wayne'的记录

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

第13.2节：使用数组的.where

任何 ActiveRecord 模型上的 where 方法都可以用来生成形如 WHERE column_name IN (a, b, c, ...) 的 SQL。这是通过传递一个数组作为参数来实现的。

作为一个简单示例，我们将使用以下模型：

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
```


Chapter 13: ActiveRecord Query Interface

ActiveRecord is the M in MVC which is the layer of the system responsible for representing business data and logic. The technique that connects the rich objects of an application to tables in a relational database management system is **O**bject **R**elational **M**apper(**ORM**).

ActiveRecord will perform queries on the database for you and is compatible with most database systems. Regardless of which database system you're using, the ActiveRecord method format will always be the same.

Section 13.1: .where

The `where` method is available on any `ActiveRecord` model and allows querying the database for a set of records matching the given criteria.

The `where` method accepts a hash where the keys correspond to the column names on the table that the model represents.

As a simple example, we will use the following model:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

To find all people with the first name of `Sven`:

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

To find all people with the first name of `Sven` and last name of `Schrodinger`:

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

In the above example, the sql output shows that records will only be returned if both the `first_name` and the `last_name` match.

query with OR condition

To find records with `first_name == 'Bruce'` OR `last_name == 'Wayne'`

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

Section 13.2: .where with an array

The `where` method on any ActiveRecord model can be used to generate SQL of the form `WHERE column_name IN (a, b, c, ...)`. This is achieved by passing an array as argument.

As a simple example, we will use the following model:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
```

```ruby
end

people = Person.where(first_name: ['马克', '玛丽'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

如果数组中包含 nil，SQL 将被修改为检查该列是否为 null：

```ruby
people = Person.where(first_name: ['马克', '玛丽', nil])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary') OR first_name IS NULL"
```

第13.3节：作用域

作用域作为 ActiveRecord 模型上的预定义过滤器。

作用域是使用 scope 类方法定义的。

作为一个简单示例，我们将使用以下模型：

```ruby
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
  #attribute :age, :integer

  # 定义一个作用域以获取所有年龄小于17岁的人
范围:未成年人, -> { 条件(年龄: 0..17) }

  # 定义一个按姓氏搜索人员的范围
scope :with_last_name, ->(name) { where(last_name: name) }

end
```

范围可以直接从模型类调用：

```ruby
minors = Person.minors
```

范围可以链式调用：

```ruby
peters_children = Person.minors.with_last_name('Peters')
```

where 方法和其他查询类型的方法也可以链式调用：

```ruby
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

在幕后，范围只是标准类方法的语法糖。例如，这些方法在功能上是相同的：

```ruby
scope :with_last_name, ->(name) { where(name: name) }

# 这行 ^ 与下面这行相同：

def self.with_last_name(name)
  where(name: name)
end
```

默认作用域

在你的模型中为所有对该模型的操作设置默认作用域。


```ruby
end

people = Person.where(first_name: ['Mark', 'Mary'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

If the array contains a `nil`, the SQL will be modified to check if the column is `null`:

```ruby
people = Person.where(first_name: ['Mark', 'Mary', nil])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary') OR first_name IS NULL"
```

Section 13.3: Scopes

Scopes act as predefined filters on `ActiveRecord` models.

A scope is defined using the `scope` class method.

As a simple example, we will use the following model:

```ruby
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
  #attribute :age, :integer

  # define a scope to get all people under 17
  scope :minors, -> { where(age: 0..17) }

  # define a scope to search a person by last name
  scope :with_last_name, ->(name) { where(last_name: name) }

end
```

Scopes can be called directly off the model class:

```ruby
minors = Person.minors
```

Scopes can be chained:

```ruby
peters_children = Person.minors.with_last_name('Peters')
```

The `where` method and other query type methods can also be chained:

```ruby
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

Behind the scenes, scopes are simply syntactic sugar for a standard class method. For example, these methods are functionally identical:

```ruby
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

Default Scope

in your model to set a default scope for all operations on the model.

> scope 方法和类方法之间有一个显著的区别：scope 定义的作用域总是会返回一个 ActiveRecord::Relation，即使其中的逻辑返回 nil。类方法则没有这样的安全保障，如果返回其他内容，可能会破坏链式调用。

第13.4节：获取第一条和最后一条记录

Rails 提供了非常简单的方法来获取数据库中的第一条和最后一条记录。

要从 users 表中获取第一条记录，我们需要输入以下命令：

```
User.first
```

它将生成以下 SQL 查询：

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` ASC LIMIT 1
```

并将返回以下记录：

```
#<User:0x007f8a6db09920 id: 1, first_name: foo, created_at: 2016年6月16日 星期四 21:43:03 UTC +00:00,
updated_at: 2016年6月16日 星期四 21:43:03 UTC +00:00 >
```

要从users表中获取最后一条记录，我们需要输入以下命令：

```
User.last
```

它将生成以下 SQL 查询：

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` DESC LIMIT 1
```

并将返回以下记录：

```
#<User:0x007f8a6db09920 id: 10, first_name: bar, created_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00,
updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

向first和last方法传递整数会创建一个LIMIT查询并返回对象数组。

```
User.first(5)
```

它将生成以下SQL查询。

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" ASC LIMIT 5
```

以及

```
User.last(5)
```

它将生成以下SQL查询。

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" DESC LIMIT 5
```


> There is one notable difference between the `scope` method and a class method: `scope`-defined scopes will *always* return an `ActiveRecord::Relation`, even if the logic within returns nil. Class methods, however, have no such safety net and can break chainability if they return something else.

Section 13.4: Get first and last record

Rails have very easy way to get `first` and `last` record from database.

To get the `first` record from `users` table we need to type following command:

```
User.first
```

It will generate following `sql` query:

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` ASC LIMIT 1
```

And will return following record:

```
#<User:0x007f8a6db09920 id: 1, first_name: foo, created_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00,
updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

To get the `last` record from `users` table we need to type following command:

```
User.last
```

It will generate following `sql` query:

```
SELECT  `users`.* FROM `users`  ORDER BY `users`.`id` DESC LIMIT 1
```

And will return following record:

```
#<User:0x007f8a6db09920 id: 10, first_name: bar, created_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00,
updated_at: Thu, 16 Jun 2016 21:43:03 UTC +00:00 >
```

Passing an integer to **first** and **last** method creates a **LIMIT** query and returns array of objects.

```
User.first(5)
```

It will generate following `sql` query.

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" ASC LIMIT 5
```

And

```
User.last(5)
```

It will generate following `sql` query.

```
SELECT  "users".* FROM "users"  ORDER BY "users"."id" DESC LIMIT 5
```

第13.5节：排序

你可以使用.order对ActiveRecord查询结果进行排序：

```
User.order(:created_at)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15
10:21:48">]
```

如果未指定，排序将默认按升序进行。你可以通过以下方式指定：

```
User.order(created_at: :asc)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15
10:21:48">]

User.order(created_at: :desc)
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13
16:51:18">]
```

.order也接受字符串，因此你也可以这样写

```
用户。排序("created_at DESC")
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13
16:51:18">]
```

由于字符串是原始SQL，你也可以指定表而不仅仅是属性。假设你想根据用户的角色名称对users进行排序，可以这样做：

```
类 User < ActiveRecord::Base
    属于:role
end

类 Role < ActiveRecord::Base
  拥有多个:users
end

User.includes(:role).order("roles.name ASC")
```

order 作用域也可以接受一个Arel节点：

```
User.includes(:role).order(User.arel_table[:name].asc)
```

第13.6节：where.not

where 条件可以使用 where.not 语法进行取反：

```
类 Person < ApplicationRecord
  #属性 :first_name, :string
结束

people = Person.where.not(first_name: ['Mark', 'Mary'])
# => SELECT "people".* FROM "people" WHERE "people"."first_name" NOT IN ('Mark', 'Mary')
```

支持 ActiveRecord 4.0 及更高版本。

Section 13.5: Ordering

You can order **ActiveRecord** query results by using `.order`:

```
User.order(:created_at)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15
10:21:48">]
```

If not specified, ordering will be performed in ascending order. You can specify it by doing:

```
User.order(created_at: :asc)
#=> => [#<User id: 2, created_at: "2015-08-12 21:36:23">, #<User id: 11, created_at: "2015-08-15
10:21:48">]

User.order(created_at: :desc)
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13
16:51:18">]
```

`.order` also accepts a string, so you could also do

```
User.order("created_at DESC")
#=> [#<User id: 7585, created_at: "2016-07-13 17:15:27">, #<User id: 7583, created_at: "2016-07-13
16:51:18">]
```

As the string is raw SQL, you can also specify a table and not only an attribute. Assuming you want to order `users` according to their `role` name, you can do this:

```
Class User < ActiveRecord::Base
    belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

The `order` scope can also accept an Arel node:

```
User.includes(:role).order(User.arel_table[:name].asc)
```

Section 13.6: where.not

`where` clauses can be negated using the `where.not` syntax:

```
class Person < ApplicationRecord
  #attribute :first_name, :string
end

people = Person.where.not(first_name: ['Mark', 'Mary'])
# => SELECT "people".* FROM "people" WHERE "people"."first_name" NOT IN ('Mark', 'Mary')
```

Supported by ActiveRecord 4.0 and later.

第13.7节：Includes

ActiveRecord 使用includes确保所有指定的关联都通过最少的查询次数加载。因此，当查询一个表的数据及其关联表时，两个表都会被加载到内存中。

```
@authors = Author.includes(:books).where(books: { bestseller: true } )

# 这将打印结果而不会额外访问数据库
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  结束
结束
```

Author.joins(:books).where(books: { bestseller: true }) 只会将符合条件的authors加载到内存中而不加载books。当不需要嵌套关联的额外信息时，使用joins。

```
@authors = Author.joins(:books).where(books: { bestseller: true } )

# 这将打印结果而不会额外查询
@authors.each { |author| puts author.name }

# 这将打印结果但会额外访问数据库
@authors.each do |author|
author.books.each do |book|
    puts book.title
  结束
结束
```

第13.8节：Joins

joins() 允许你将表连接到当前模型。例如。

```
用户。连接(:帖子)
```

将生成以下SQL查询：

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id""
```

连接表后，您将可以访问它：

```
用户。连接(:帖子).条件(帖子: { 标题: "Hello world" })
```

注意复数形式。如果您的关联是:has_many，那么joins()的参数应使用复数形式。
否则，使用单数形式。

嵌套连接：

```
用户。连接(帖子: :图片).条件(图片: { 标题说明: '第一篇帖子' })
```

这将生成：

```
SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER JOIN
"images" ON "images"."post_id" = "images"."id"
```


Section 13.7: Includes

ActiveRecord with `includes` ensures that all of the specified associations are loaded using the minimum possible number of queries. So when querying a table for data with an associated table, both tables are loaded into memory.

```
@authors = Author.includes(:books).where(books: { bestseller: true } )

# this will print  results without additional db hitting
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

`Author.joins(:books).where(books: { bestseller: true } )` will load only **authors** with conditions into memory **without loading books**. Use `joins` when additional information about nested associations isn't required.

```
@authors = Author.joins(:books).where(books: { bestseller: true } )

# this will print results without additional queries
@authors.each { |author| puts author.name }

# this will print results with additional db queries
@authors.each do |author|
  author.books.each do |book|
    puts book.title
  end
end
```

Section 13.8: Joins

`joins()` allows you to join tables to your current model. For ex.

```
User.joins(:posts)
```

will produce the following SQL query:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id""
```

Having table joined, you will have access to it:

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

Pay attention on plural form. If your relation is `:has_many`, then the `joins()` argument should be pluralized. Otherwise, use singular.

Nested `joins`:

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

which will produce:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER JOIN
"images" ON "images"."post_id" = "images"."id""
```

第13.9节：限制和偏移

你可以使用limit来指定要获取的记录数，使用offset来指定在开始返回记录之前要跳过的记录数。

例如

```
User.limit(3) #返回前三条记录
```

它将生成以下SQL查询。

```
SELECT  `users`.* FROM `users` LIMIT 3
```

由于上述查询中未提及offset，因此它将返回前三条记录。

```
User.limit(5).offset(30) #返回从第31条开始的5条记录，即第31条到第35条
```

它将生成以下SQL查询。

```
SELECT  `users`.* FROM `users` LIMIT 5 OFFSET 30
```

第13.10节：.find_by

您可以使用find_by通过表中的任何字段查找记录。

所以，如果你有一个带有first_name属性的User模型，你可以这样做：

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

请注意，find_by默认不会抛出任何异常。如果结果是空集，它会返回 nil，而不是find。

如果需要抛出异常，可以使用find_by!，它会像find一样抛出ActiveRecord::RecordNotFound错误。

第13.11节：.delete_all

如果你需要快速删除大量记录，**ActiveRecord**提供了.delete_all方法。可以直接在模型上调用，以删除该表中的所有记录，或删除一个集合。但要注意，.delete_all不会实例化任何对象，因此不会触发任何回调（before_*和after_destroy不会被触发）。

```
User.delete_all
#=> 39  <-- .delete_all 返回删除的行数

User.where(name: "John").delete_all
```

第13.12节：ActiveRecord 不区分大小写的搜索

如果你需要在 ActiveRecord 模型中搜索相似的值，你可能会想使用LIKE或ILIKE，但这在不同的数据库引擎之间不可移植。同样，总是将字符串转换为小写或大写也可能导致性能问题。

你可以使用 ActiveRecord 底层的 Arel 的matches方法以安全的方式实现这一点：

Section 13.9: Limit and Offset

You can use `limit` to tell the number of records to be fetched, and use `offset` to tell the number of records to skip before starting to return the records.

For Example

```
User.limit(3) #returns first three records
```

It will generate following sql query.

```
"SELECT  `users`.* FROM `users` LIMIT 3"
```

As offset is not mentioned in above query so it will return first three records.

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

It will generate following sql query.

```
"SELECT  `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

Section 13.10: .find_by

You can find records by any field in your table using `find_by`.

So, if you have a `User` model with a `first_name` attribute you can do:

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

Mind that `find_by` doesn't throw any exception by default. If the result is an empty set, it returns **`nil`** instead of `find`.

If the exception is needed may use `find_by!` that raises an **`ActiveRecord::RecordNotFound`** error like `find`.

Section 13.11: .delete_all

If you need to delete a lot of records quickly, **ActiveRecord** gives **`.delete_all`** method. to be called directly on a model, to delete all records in that table, or a collection. Beware though, as **`.delete_all`** does not instantiate any object hence does not provide any callback (`before_*` and `after_destroy` don't get triggered).

```
User.delete_all
#=> 39  <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

Section 13.12: ActiveRecord case insensitive search

If you need to search an ActiveRecord model for similar values, you might be tempted to use `LIKE` or `ILIKE` but this isn't portable between database engines. Similarly, resorting to always downcasing or upcasing can create performance issues.

You can use ActiveRecord's underlying Arel `matches` method to do this in a safe way:

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Arel 会根据配置的数据库引擎应用适当的 LIKE 或 ILIKE 结构。

第13.13节：.group 和 .count

我们有一个Product模型，想要按它们的category进行分组。

```
Product.select(:category).group(:category)
```

这将向数据库发出如下查询：

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

确保分组字段也被选中。分组对于计数——在本例中——类别的出现尤其有用。

```
Product.select(:category).group(:category).count
```

正如查询所示，它将使用数据库进行计数，这比先检索所有记录然后在代码中计数要高效得多：

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS products_category
FROM "products" GROUP BY "products"."category"
```

第13.14节：.distinct（或 .uniq）

如果你想从结果中去除重复项，可以使用.distinct()：

```
Customers.select(:country).distinct
```

这会向数据库发出如下查询：

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

.uniq() 具有相同的效果。在Rails 5.0中它被弃用，并将在Rails 5.1版本中移除。原因是，单词unique的含义与distinct不同，可能会引起误解。此外
distinct更接近SQL语法。


```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Arel will apply the appropriate LIKE or ILIKE construct for the database engine configured.

Section 13.13: .group and .count

We have a `Product` model and we want to group them by their `category`.

```
Product.select(:category).group(:category)
```

This will query the database as follows:

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

Make sure that the grouped field is also selected. Grouping is especially useful for counting the occurrence - in this case - of `categories`.

```
Product.select(:category).group(:category).count
```

As the query shows, it will use the database for counting, which is much more efficient, than retrieving all record first and do the counting in the code:

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS products_category
FROM "products" GROUP BY "products"."category"
```

Section 13.14: .distinct (or .uniq)

If you want to remove duplicates from a result, you can use `.distinct()`:

```
Customers.select(:country).distinct
```

This queries the database as follows:

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()` has the same effect. With Rails 5.0 it got deprecated and it will be removed from Rails with version 5.1. The reason is, that the word `unique` doesn't have the same meaning as distinct and it can be misleading. Furthermore `distinct` is closer to the SQL syntax.

第14章：ActionMailer

Action Mailer 允许你使用邮件类和视图从应用程序发送电子邮件。邮件类的工作方式与控制器非常相似。它们继承自 ActionMailer::Base，位于 app/mailers 目录中，并且有对应的视图文件，位于 app/views 中。

第14.1节：基础邮件类

此示例使用了四个不同的文件：

- User 模型
- User 邮件类
- 电子邮件的 html 模板
- 电子邮件的纯文本模板

在这种情况下，User 模型调用邮件类中的approved方法，并传递已批准的post（模型中的approved方法可能由回调、控制器方法等调用）。然后，邮件类使用传入的post中的信息（例如标题）从 html 或纯文本模板生成电子邮件。

默认情况下，邮件类使用与方法同名的模板（这就是为什么邮件类方法和模板都叫“approved”的原因）。

user_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "donotreply@example.com"

  def approved(post)
      @title = post.title
      @user = post.user
mail(to: @user.email, subject: "Your Post was Approved!")
  end
end
```

user.rb

```
def approved(post)
    UserMailer.approved(post)
end
```

approved.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>帖子已通过审核</title>
    </head>
    <body>
        <h2>恭喜<%= @user.name %>！您的帖子（#<%= @title %>）已通过审核！</h2>
        <p>期待您未来的更多帖子！</p>
    </body>
</html>
```

approved.text.erb

```
恭喜 <%= @user.name %>！您的帖子（#<%= @title %>）已通过审核！
期待您未来的更多帖子！
```


Chapter 14: ActionMailer

Action Mailer allows you to send emails from your application using mailer classes and views. Mailers work very similarly to controllers. They inherit from ActionMailer::Base and live in app/mailers, and they have associated views that appear in app/views.

Section 14.1: Basic Mailer

This example uses four different files:

- The User model
- The User mailer
- The html template for the email
- The plain-text template for the email

In this case, the user model calls the `approved` method in the mailer and passes the `post` that has been approved (the `approved` method in the model may be called by a callback, from a controller method, etc). Then, the mailer generates the email from either the html or plain-text template using the information from the passed-in `post` (e.g. the title). By default, the mailer uses the template with the same name as the method in the mailer (which is why both the mailer method and the templates have the name 'approved').

user_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "donotreply@example.com"

  def approved(post)
      @title = post.title
      @user = post.user
      mail(to: @user.email, subject: "Your Post was Approved!")
  end
end
```

user.rb

```
def approved(post)
    UserMailer.approved(post)
end
```

approved.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <title>Post Approved</title>
    </head>
    <body>
        <h2>Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!</h2>
        <p>We look forward to your future posts!</p>
    </body>
</html>
```

approved.text.erb

```
Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!
We look forward to your future posts!
```

第14.2节：生成新的邮件发送器

要生成新的邮件发送器，请输入以下命令

```
rails generate mailer PostMailer
```

这将在app/mailers/post_mailer.rb中生成一个名为PostMailer的空白模板文件

```
class PostMailer < ApplicationMailer
end
```

还将为电子邮件视图生成两个布局文件，一个用于html格式，一个用于文本格式。

如果您不想使用生成器，可以自己创建邮件发送器。确保它们继承自
`ActionMailer::Base`

第14.3节：ActionMailer 拦截器

Action Mailer 提供了拦截器方法的钩子。这些允许您注册在邮件发送生命周期中被调用的类。

拦截器类必须实现 :delivering_email(message) 方法，该方法将在邮件发送之前被调用，允许您在邮件到达发送代理之前对其进行修改。您的类应直接对传入的 Mail::Message 实例进行任何必要的修改。

对于开发者来说，将邮件发送给自己而非真实用户可能很有用。

注册 actionmailer 拦截器的示例：

```
# config/initializers/override_mail_recipient.rb

if Rails.env.development? or Rails.env.test?
   class OverrideMailRecipient
      def self.delivering_email(mail)
        mail.subject = '这是虚拟主题'
        mail.bcc = 'test_bcc@noemail.com'
mail.to = 'test@noemail.com'
      end
   end
   ActionMailer::Base.register_interceptor(OverrideMailRecipient)
end
```

第14.4节：添加附件

ActionMailer 也允许添加附件。

```
attachments['filename.jpg'] = File.read('/path/to/filename.jpg')
```

默认情况下，附件将使用Base64编码。要更改此设置，可以向attachments方法添加一个哈希。

```
attachments['filename.jpg'] = {
  mime_type: 'application/gzip',
  encoding: 'SpecialEncoding',
  content: encoded_content
```


Section 14.2: Generating a new mailer

To generate a new mailer, enter the following command

```
rails generate mailer PostMailer
```

This will generate a blank template file in `app/mailers/post_mailer.rb` named *PostMailer*

```
class PostMailer < ApplicationMailer
end
```

Two layout files will also be generated for the email view, one for the html format and one for the text format.

If you prefer not to use the generator, you can create your own mailers. Make sure they inherit from
`ActionMailer::Base`

Section 14.3: ActionMailer Interceptor

Action Mailer provides hooks into the interceptor methods. These allow you to register classes that are called during the mail delivery life cycle.

An interceptor class must implement the :delivering_email(message) method which will be called before the email is sent, allowing you to make modifications to the email before it hits the delivery agents. Your class should make any needed modifications directly to the passed in Mail::Message instance.

It can be useful for developers to send email to themselves not real users.

Example of registering an actionmailer interceptor:

```
# config/initializers/override_mail_recipient.rb

if Rails.env.development? or Rails.env.test?
  class OverrideMailRecipient
    def self.delivering_email(mail)
      mail.subject = 'This is dummy subject'
      mail.bcc = 'test_bcc@noemail.com'
      mail.to = 'test@noemail.com'
    end
  end
  ActionMailer::Base.register_interceptor(OverrideMailRecipient)
end
```

Section 14.4: Adding Attachments

`ActionMailer` also allows attaching files.

```
attachments['filename.jpg'] = File.read('/path/to/filename.jpg')
```

By default, attachments will be encoded with `Base64`. To change this, you can add a hash to the attachments method.

```
attachments['filename.jpg'] = {
  mime_type: 'application/gzip',
  encoding: 'SpecialEncoding',
  content: encoded_content
```

```
}
```

你也可以添加内联附件

```
attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')
```

第14.5节：ActionMailer回调

ActionMailer支持三种回调

- before_action
- after_action
- around_action

在你的Mailer类中提供这些回调

```
class UserMailer < ApplicationMailer
after_action :set_delivery_options, :prevent_delivery_to_guests, :set_business_headers
```

然后在private关键字下创建这些方法

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

第14.6节：生成定期通讯

创建Newsletter模型：

```
rails g model Newsletter name:string email:string

  subl app/models/newsletter.rb

validates :name, presence: true
  validates :email, presence: true
```

创建Newsletter控制器：

```
  rails g controller Newsletters create

  class NewslettersController < ApplicationController
    skip_before_action :authenticate_user!
before_action :set_newsletter, only: [:destroy]

    def create
      @newsletter = Newsletter.create(newsletter_params)
      if @newsletter.save
        redirect_to blog_index_path
      else
```



```
}
```

You can also add inline attachments

```
attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')
```

Section 14.5: ActionMailer Callbacks

ActionMailer supports three callbacks

- before_action
- after_action
- around_action

Provide these in your Mailer class

```
class UserMailer < ApplicationMailer
  after_action :set_delivery_options, :prevent_delivery_to_guests, :set_business_headers
```

Then create these methods under the `private` keyword

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

Section 14.6: Generate a Scheduled Newsletter

Create the **Newsletter** model:

```
  rails g model Newsletter name:string email:string

  subl app/models/newsletter.rb

  validates :name, presence: true
  validates :email, presence: true
```

Create the **Newsletter** controller:

```
  rails g controller Newsletters create

  class NewslettersController < ApplicationController
    skip_before_action :authenticate_user!
    before_action :set_newsletter, only: [:destroy]

    def create
      @newsletter = Newsletter.create(newsletter_params)
      if @newsletter.save
        redirect_to blog_index_path
      else
```

```
        redirect_to root_path
      结束
    结束

    private

      def set_newsletter
        @newsletter = Newsletter.find(params[:id])
      end

      def newsletter_params
params.require(:newsletter).permit(:name, :email)
      end

  end
```

之后，将create.html.erb视图更改为新名称。我们将把此文件转换为partial view，存储在Footer中。名称将为_form.html.erb。

更改文件名从： app/views/newsletters/create.html.erb **到：** app/views/newsletters/_form.html.erb

之后设置路由：

```
subl app/config/routes.rb

  resources :newsletters
```

接下来，我们需要设置用于保存每个邮件的表单：

```
subl app/views/newsletters/_form.html.erb

  <%= form_for (Newsletter.new) do |f| %>
    <div class="col-md-12" style="margin: 0 auto; padding: 0;">
      <div class="col-md-6" style="padding: 0;">
        <%= f.text_field :name, class: 'form-control', placeholder:'Nombre' %>
      </div>
      <div class="col-md-6" style="padding: 0;">
        <%= f.text_field :email, class: 'form-control', placeholder:'Email' %>
      </div>
    </div>
    <div class="col-md-12" style="margin: 0 auto; padding:0;">
      <%= f.submit class:"col-md-12 tran3s s-color-bg hvr-shutter-out-horizontal", style:'border:
none; color: white; cursor: pointer; margin: 0.5em auto; padding: 0.75em; width: 100%;' %>
    </div>
  <% end %>
```

然后在页脚插入：

```
subl app/views/layouts/_footer.html.erb

  <%= render 'newsletters/form' %>
```

现在，安装 -letter_opener-，可以在默认浏览器中预览邮件，而不是发送邮件。这意味着你不需要在开发环境中设置邮件发送，也不必担心意外将测试邮件发送给其他人。


```
        redirect_to root_path
      end
    end

    private

      def set_newsletter
        @newsletter = Newsletter.find(params[:id])
      end

      def newsletter_params
        params.require(:newsletter).permit(:name, :email)
      end

  end
```

After that, change the **create.html.erb** view to the nex name. We will convert this file to and **partial view** which will be stored inside the **Footer**. The name will be **_form.html.erb**.

Change name file from: app/views/newsletters/create.html.erb **To:** app/views/newsletters/_form.html.erb

After that set the routes:

```
subl app/config/routes.rb

resources :newsletters
```

Later on, we need to set the form we will use to save each mail:

```
subl app/views/newsletters/_form.html.erb

<%= form_for (Newsletter.new) do |f| %>
  <div class="col-md-12" style="margin: 0 auto; padding: 0;">
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :name, class: 'form-control', placeholder:'Nombre' %>
    </div>
    <div class="col-md-6" style="padding: 0;">
      <%= f.text_field :email, class: 'form-control', placeholder:'Email' %>
    </div>
  </div>
  <div class="col-md-12" style="margin: 0 auto; padding:0;">
    <%= f.submit class:"col-md-12 tran3s s-color-bg hvr-shutter-out-horizontal", style:'border:
none; color: white; cursor: pointer; margin: 0.5em auto; padding: 0.75em; width: 100%;' %>
  </div>
<% end %>
```

And after that, insert on the footer:

```
subl app/views/layouts/_footer.html.erb

<%= render 'newsletters/form' %>
```

Now, install the **-letter_opener-** to can preview email in the default browser instead of sending it. This means you do not need to set up email delivery in your development environment, and you no longer need to worry about accidentally sending a test email to someone else's address.

首先将该 gem 添加到你的开发环境中，然后运行 bundle 命令进行安装。

```
subl your_project/Gemfile

  gem "letter_opener", :group => :development
```

然后在开发环境中设置发送方式：

```
subl your_project/app/config/environments/development.rb

  config.action_mailer.delivery_method = :letter_opener
```

现在，创建一个Mailer结构来管理我们将要使用的所有邮件。在终端中执行

```
  rails generate mailer UserMailer newsletter_mailer
```

在UserMailer中，我们需要创建一个名为Newsletter Mailer的方法，该方法将包含最新的博客文章内容，并通过rake任务触发。我们假设你之前已经创建了博客结构。

```
subl your_project/app/mailers/user_mailer.rb

class UserMailer 'your_gmail_account@gmail.com'


  def newsletter_mailer
    @newsletter = Newsletter.all
    @post = Post.last(3)
emails = @newsletter.collect(&:email).join(", ")
    mail(to: emails, subject: "Hi, this is a test mail.")
  end

end
```

之后，创建邮件模板：

```
subl your_project/app/views/user_mailer/newsletter_mailer.html.erb

<p> 亲爱的关注者： </p>
<p> 这些是我们博客的最新文章。我们邀请您阅读并分享我们本周所做的一切。
</p>

<br/>
<table>
<% @post.each do |post| %>
  <%#= link_to blog_url(post) do %>
      <tr style="display:flex; float:left; clear:both;">
        <td style="display:flex; float:left; clear:both; height: 80px; width: 100px;">
          <% if post.cover_image.present? %>
            <%= image_tag post.cover_image.fullsize.url, class:"principal-home-image-slider" %>
          <%# else %>
            <%#= image_tag 'http://your_site_project.com' + post.cover_video, class:"principal-home-image-slider" %>
            <%#= raw(video_embed(post.cover_video)) %>
          <% end %>
```


First add the gem to your development environment and run the bundle command to install it.

```
  subl your_project/Gemfile

  gem "letter_opener", :group => :development
```

Then set the delivery method in the Development Environment:

```
  subl your_project/app/config/environments/development.rb

  config.action_mailer.delivery_method = :letter_opener
```

Now, create an **Mailer Structure** to manage the whole mailers which we will work. In terminal

```
  rails generate mailer UserMailer newsletter_mailer
```

And inside the **UserMailer**, we have to create a method called **Newsletter Mailer** which will be created to contain inside on the lastest blog post and will be fired with a rake action. We will assume that you had a blog structure created before.

```
subl your_project/app/mailers/user_mailer.rb

class UserMailer 'your_gmail_account@gmail.com'


  def newsletter_mailer
    @newsletter = Newsletter.all
    @post = Post.last(3)
    emails = @newsletter.collect(&:email).join(", ")
    mail(to: emails, subject: "Hi, this is a test mail.")
  end

end
```

After that, create the **Mailer Template**:

```
subl your_project/app/views/user_mailer/newsletter_mailer.html.erb

<p> Dear Followers: </p>
<p> Those are the lastest entries to our blog. We invite you to read and share everything we did on
this week. </p>

<br/>
<table>
<% @post.each do |post| %>
  <%#= link_to blog_url(post) do %>
      <tr style="display:flex; float:left; clear:both;">
        <td style="display:flex; float:left; clear:both; height: 80px; width: 100px;">
          <% if post.cover_image.present? %>
            <%= image_tag post.cover_image.fullsize.url, class:"principal-home-image-slider" %>
          <%# else %>
            <%#= image_tag 'http://your_site_project.com' + post.cover_video, class:"principal-home-image-slider" %>
            <%#= raw(video_embed(post.cover_video)) %>
          <% end %>
```

```
        </td>
        <td>
          <h3>
            <%= link_to post.title, :controller => "blog", :action => "show", :only_path => false,
:id => post.id %>
          </h3>
          <p><%= post.subtitle %></p>
        </td>
        <td style="display:flex; float:left; clear:both;">

        </td>
      </tr>
  <%# end %>
<% end %>
</table>
```

由于我们希望将发送邮件作为一个独立的进程，因此让我们创建一个 Rake 任务来触发发送邮件。向你的 Rails 应用程序的 lib/tasks 目录添加一个名为 email_tasks.rake 的新文件：

```
touch lib/taks/email_tasks.rake

desc 'weekly newsletter email'
task weekly_newsletter_email: :environment do
  UserMailer.newsletter_mailer.deliver!
end
```

send_digest_email: :environment 表示在运行任务之前加载 Rails 环境，这样你就可以在任务中访问应用程序的类（如 UserMailer）。

现在，运行命令 rake -T 会列出新创建的 Rake 任务。通过运行该任务并检查邮件是否发送来测试一切是否正常。

要测试邮件发送方法是否有效，运行以下 rake 命令：

```
  rake weekly_newsletter_email
```

此时，我们已有一个可用的 rake 任务，可以使用 **crontab** 进行调度。因此我们将安装 **Whenever Gem**，该 Gem 用于提供编写和部署 cron 任务的简洁语法。

```
subl your_project/Gemfile

  gem 'whenever', require: false
```

之后，运行下一条命令为你创建一个初始的 config/schedule.rb 文件（前提是你的项目中已存在 config 文件夹）。

```
wheneverize。

  【add】写入 `./config/schedule.rb`
[完成] wheneverized !
```

现在，在调度文件中，我们必须创建我们的**CRON 任务**，并在确定
CRON 任务时调用邮件发送方法，以便在选定的时间范围内无需辅助地执行某些任务。你可以使用不同类型的语法，详见此链接。


```
        </td>
        <td>
          <h3>
            <%= link_to post.title, :controller => "blog", :action => "show", :only_path => false,
:id => post.id %>
          </h3>
          <p><%= post.subtitle %></p>
        </td>
        <td style="display:flex; float:left; clear:both;">

        </td>
      </tr>
  <%# end %>
<% end %>
</table>
```

Since we want to send the email as a separate process, let's create a Rake task to fire off the email. Add a new file called email_tasks.rake to lib/tasks directory of your Rails application:

```
touch lib/taks/email_tasks.rake

desc 'weekly newsletter email'
task weekly_newsletter_email: :environment do
  UserMailer.newsletter_mailer.deliver!
end
```

The send_digest_email: :environment means to load the Rails environment before running the task, so you can access the application classes (like UserMailer) within the task.

Now, running the command rake -T will list the newly created Rake task. Test everything works by running the task and checking whether the email is sent or not.

To test if the mailer method works, run the rake command:

```
  rake weekly_newsletter_email
```

At this point, we have a working rake task which can be scheduled using **crontab**. So we will install the **Whenever Gem** which is used to provide a clear syntax for writing and deploying cron jobs.

```
subl your_project/Gemfile

  gem 'whenever', require: false
```

After that, run the next command to create an initial config/schedule.rb file for you (as long as the config folder is already present in your project).

```
  wheneverize .

  【add】 writing `./config/schedule.rb'
  [done] wheneverized!
```

Now, inside the schedule file, we have to create our **CRON JOB** and call the mailer method inside determining the CRON JOB to operate some tasks without assistance and in a selected range of time. You can use different types of syntax as is explained on this link.

```
subl your_project/config/schedule.rb

every 1.day, :at => '4:30 am' do
  rake 'weekly_newsletter_email'
end
```


```
every 3.hours do # 1.minute 1.day 1.week 1.month 1.year is also supported
  runner "MyModel.some_process"
  rake "my:rake:task"
  command "/usr/bin/my_great_command"
end

every 1.day, :at => '4:30 am' do
  runner "MyModel.task_to_run_at_four_thirty_in_the_morning"
end

every :hour do # Many shortcuts available: :hour, :day, :month, :year, :reboot
  runner "SomeModel.ladeeda"
end

every :sunday, :at => '12pm' do # Use any day of the week or :weekend, :weekday
  runner "Task.do_something_great"
end

every '0 0 27-31 * *' do
  command "echo 'you can use raw cron syntax too'"
end

# run this task only on servers with the :app role in Capistrano
# see Capistrano roles section below
every :day, :at => '12:20am', :roles => [:app] do
  rake "app_server:task"
end
```

现在，为了测试**Cron 任务**是否成功创建，我们可以使用以下命令从终端读取我们的计划任务（CRON 语法）：

```
your_project your_mac_user$ whenever
```


subl your_project/config/schedule.rb

every 1.day, :at => '4:30 am' do
  rake 'weekly_newsletter_email'
end

every 3.hours do # 1.minute 1.day 1.week 1.month 1.year is also supported
  runner "MyModel.some_process"
  rake "my:rake:task"
  command "/usr/bin/my_great_command"
end

every 1.day, :at => '4:30 am' do
  runner "MyModel.task_to_run_at_four_thirty_in_the_morning"
end

every :hour do # Many shortcuts available: :hour, :day, :month, :year, :reboot
  runner "SomeModel.ladeeda"
end

every :sunday, :at => '12pm' do # Use any day of the week or :weekend, :weekday
  runner "Task.do_something_great"
end

every '0 0 27-31 * *' do
  command "echo 'you can use raw cron syntax too'"
end

run this task only on servers with the :app role in Capistrano
see Capistrano roles section below
every :day, :at => '12:20am', :roles => [:app] do
  rake "app_server:task"
end

Now to test the **Cron Job** was succesfully created we can use the next command to read since terminal, our scheduled job in CRON SYNTAX:

your_project your_mac_user$ whenever

```
30 4 * * * /bin/bash -l -c 'cd /Users/your_mac_user/Desktop/your_project && RAILS_ENV=production
bundle exec rake weekly_newsletter_email --silent'
```

现在，为了在开发环境中运行测试，建议在**application.rb**主文件中设置以下行，
让应用程序知道它将使用哪些模型。

```
subl your_project/config/application.rb

  config.action_mailer.default_url_options = { :host => "http://localhost:3000/" }
```

现在，为了让Capistrano V3在服务器内保存新的Cron Job以及触发该任务执行的触发器，我们需要添加以下要求：

```
subl your_project/Capfile

  require 'whenever/capistrano'
```

并将**deploy**文件中插入**CRON JOB**将使用的关于**environment**和
application名称的标识符。

```
subl your_project/config/deploy.rb

set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

完成后，保存每个文件的更改，运行capistrano部署命令：

```
  cap production deploy
```

现在你的JOB已经创建并安排运行Mailer方法，这正是我想要的，并且在我们在这些文件中设置的时间范围内执行。


```
30 4 * * * /bin/bash -l -c 'cd /Users/your_mac_user/Desktop/your_project && RAILS_ENV=production
bundle exec rake weekly_newsletter_email --silent'
```

Now, to run the test in Development Environment, is wise to set the next line on the **application.rb** principal file to let the application knows where are the models it will use.

```
  subl your_project/config/application.rb

  config.action_mailer.default_url_options = { :host => "http://localhost:3000/" }
```

Now to let **Capistrano V3** save the new **Cron Job** inside the server and the trigger which will fired up the execution of this task, we have to add the next requirement:

```
  subl your_project/Capfile

  require 'whenever/capistrano'
```

And insert into the **deploy** file the identifier which **CRON JOB** will use about the **environment** and the name of the **application**.

```
subl your_project/config/deploy.rb

set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

And ready, after save changes on each file, run the capistrano deploy command:

```
  cap production deploy
```

And now your JOB was created and calendarize to run the Mailer Method which is what i want and in the range of time we set on this files.

第15章：Rails生成命令

参数	详情
-h/--help	获取任何生成器命令的帮助
-p/--pretend	假装模式：运行生成器但不会创建或更改任何文件
field:type	'field-name' 是要创建的列名，'type' 是列的数据类型。'field:type' 中 'type' 的可能取值在备注部分给出。

用法：rails generate GENERATOR_NAME [args] [options]。

使用 rails generate 列出可用的生成器。别名：rails g。

第15.1节：Rails生成控制器

我们可以使用 rails g controller 命令创建一个新的控制器。

```
$ bin/rails generate controller controller_name
```

控制器生成器期望的参数格式为generate controller ControllerName action1 action2。

以下命令创建一个名为 Greetings 的控制器，并带有一个 hello 动作。

```
$ bin/rails generate controller Greetings hello
```

你将看到以下输出

```
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/controllers/greetings_controller_test.rb
     invoke  helper
create    app/helpers/greetings_helper.rb
     invoke  assets
      invoke    coffee
创建      app/assets/javascripts/greetings.coffee
     调用    scss
创建      app/assets/stylesheets/greetings.scss
```

这将生成以下内容

文件	示例
控制器文件	greetings_controller.rb
视图文件	hello.html.erb
功能测试文件	greetings_controller_test.rb
视图助手	greetings_helper.rb
JavaScript 文件	问候。咖啡

它还会为 routes.rb 中的每个操作添加路由。

Chapter 15: Rails generate commands

Parameter	Details
-h/--help	Get help on any generator command
-p/--pretend	Pretend Mode: Run generator but will not create or change any files
field:type	'field-name' is the name of the column to be created and 'type' is the data-type of column. The possible values for 'type' in field:type are given in the Remarks section.

Usage: rails generate GENERATOR_NAME [args] [options].

Use rails generate to list available generators. Alias: rails g.

Section 15.1: Rails Generate Controller

we can create a new controller with rails g controller command.

```
$ bin/rails generate controller controller_name
```

The controller generator is expecting parameters in the form of generate controller ControllerName action1 action2.

The following creates a Greetings controller with an action of hello.

```
$ bin/rails generate controller Greetings hello
```

You will see the following output

```
    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/controllers/greetings_controller_test.rb
    invoke  helper
    create    app/helpers/greetings_helper.rb
    invoke  assets
    invoke    coffee
    create      app/assets/javascripts/greetings.coffee
    invoke    scss
    create      app/assets/stylesheets/greetings.scss
```

This generates the following

File	Example
Controller File	greetings_controller.rb
View File	hello.html.erb
Functional Test File	greetings_controller_test.rb
View Helper	greetings_helper.rb
JavaScript File	greetings.coffee

It will also add routes for each action in routes.rb

第15.2节：Rails 生成迁移

你可以使用以下命令从终端生成 Rails 迁移文件：

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

要查看该命令支持的所有选项列表，可以像下面这样不带任何参数运行该命令
`rails generate migration`.

例如，如果你想向 `users` 表添加 `first_name` 和 `last_name` 字段，可以这样做：

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails 将创建以下迁移文件：

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

现在，通过在终端运行以下命令，将待处理的迁移应用到数据库：

版本 < 5.0

```
rake db:migrate
```

版本 ≥ 5.0

```
rails db:migrate
```

> 注意： 为了减少输入，你可以将 generate 替换为 g。

第15.3节：Rails 生成脚手架

免责声明：除非是非常常规的CRUD应用/测试，否则不推荐使用脚手架。这可能会生成大量不需要的文件（视图/模型/控制器），从而给你的网络应用带来麻烦（糟糕 :()。

要为新对象生成一个完整可用的脚手架，包括模型、控制器、视图、资源和测试，请使用
`rails g scaffold` 命令。

```
$ rails g scaffold Widget name:string price:decimal
    invoke  active_record
create    db/migrate/20160722171221_create_widgets.rb
    create    app/models/widget.rb
调用    test_unit
    创建      test/models/widget_test.rb
    创建      test/fixtures/widgets.yml
    调用  resource_route
路由    资源 :widgets
    调用  scaffold_controller
    创建    app/controllers/widgets_controller.rb
    调用    erb
创建      app/views/widgets
    创建      app/views/widgets/index.html.erb
    创建      app/views/widgets/edit.html.erb
```


Section 15.2: Rails Generate Migration

You can generate a rails migration file from the terminal using the following command:

```
rails generate migration NAME [field[:type][:index] field[:type][:index]] [options]
```

For a list of all the options supported by the command, you could run the command without any arguments as in `rails generate migration`.

For example, if you want to add `first_name` and `last_name` fields to `users` table, you can do:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails will create the following migration file:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

Now, apply the pending migrations to the database by running the following in the terminal:

Version < 5.0

```
rake db:migrate
```

Version ≥ 5.0

```
rails db:migrate
```

> **Note:** For even less typing, you can replace `generate` with `g`.

Section 15.3: Rails Generate Scaffold

DISCLAIMER: Scaffolding is not recommended unless it's for very conventional CRUD apps/testing. This may generate a lot of files(views/models/controllers) that are not needed in your web application thus causing headaches(bad :().

To generate a fully working scaffold for a new object, including model, controller, views, assets, and tests, use the `rails g scaffold` command.

```
$ rails g scaffold Widget name:string price:decimal
    invoke  active_record
    create    db/migrate/20160722171221_create_widgets.rb
    create    app/models/widget.rb
    invoke    test_unit
    create      test/models/widget_test.rb
    create      test/fixtures/widgets.yml
    invoke  resource_route
     route    resources :widgets
    invoke  scaffold_controller
    create    app/controllers/widgets_controller.rb
    invoke    erb
    create      app/views/widgets
    create      app/views/widgets/index.html.erb
    create      app/views/widgets/edit.html.erb
```

```
创建      app/views/widgets/show.html.erb
    创建      app/views/widgets/new.html.erb
    创建      app/views/widgets/_form.html.erb
    调用    test_unit
创建      test/controllers/widgets_controller_test.rb
    调用    helper
创建      app/helpers/widgets_helper.rb
    调用    jbuilder
创建      app/views/widgets/index.json.jbuilder
    创建      app/views/widgets/show.json.jbuilder
    调用  assets
调用    javascript
创建      app/assets/javascripts/widgets.js
    调用    scss
创建      app/assets/stylesheets/widgets.scss
```

然后你可以运行 `rake db:migrate` 来设置数据库表。

然后你可以访问 http://localhost:3000/widgets ，你将看到一个功能齐全的 CRUD 脚手架。

第15.4节：Rails 生成模型

要生成一个 ActiveRecord 模型，该模型会自动创建正确的数据库迁移和样板测试文件，输入以下命令

```
rails generate model NAME column_name:column_type
```

“NAME”是模型的名称。“field”是数据库表中列的名称，“type”是列的类型（例如 name:string 或 body:text）。请查看备注部分以获取支持的列类型列表。

要设置外键，添加 belongs_to:model_name。

假设你想设置一个 User 模型，该模型有一个 username、email 并且属于一个 School，你可以输入以下命令

```
rails generate model User username:string email:string school:belongs_to
```

rails g 是 rails generate 的简写。这将产生相同的结果

```
rails g model User username:string email:string school:belongs_to
```



```
    create      app/views/widgets/show.html.erb
    create      app/views/widgets/new.html.erb
    create      app/views/widgets/_form.html.erb
    invoke    test_unit
    create      test/controllers/widgets_controller_test.rb
    invoke    helper
    create      app/helpers/widgets_helper.rb
    invoke    jbuilder
    create      app/views/widgets/index.json.jbuilder
    create      app/views/widgets/show.json.jbuilder
    invoke  assets
    invoke    javascript
    create      app/assets/javascripts/widgets.js
    invoke    scss
    create      app/assets/stylesheets/widgets.scss
```

Then you can run `rake db:migrate` to set up the database table.

Then you can visit http://localhost:3000/widgets and you'll see a fully functional CRUD scaffold.

Section 15.4: Rails Generate Model

To generate an `ActiveRecord` model that automagically creates the correct db migrations & boilerplate test files for your model, enter this command

```
rails generate model NAME column_name:column_type
```

'NAME' is the name of the model. 'field' is the name of the column in the DB table and 'type' is the column type (e.g. `name:string` or `body:text`). Check the Remarks section for a list of supported column types.

To setup foreign keys, add `belongs_to:model_name`.

So say you wanted to setup a `User` model that has a `username`, `email` and belongs to a `School`, you would type in the following

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` is shorthand for `rails generate`. This would produce the same result

```
rails g model User username:string email:string school:belongs_to
```

第16章：配置

第16.1节：自定义配置

在**config**/目录下创建一个**YAML**文件，例如：`config/neo4j.yml`

`neo4j.yml`的内容可以如下（为简化起见，所有环境均使用`default`）：

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

在config/application.rb中：

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

现在，您的自定义配置可以通过以下方式访问：

```
Rails.configuration.neo4j['host']
 #=> localhost
Rails.configuration.neo4j['port']
 #=> 7474
```

更多信息

Rails 官方 API 文档中对 config_for 方法的描述是：

> 方便加载当前 Rails 环境下的 config/foo.yml 配置文件。

如果您不想使用 yaml 文件

您可以通过 Rails 配置对象，在

`config.x` 属性下使用自定义配置来配置您自己的代码。

示例

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.x.super_debugger = true
```


Chapter 16: Configuration

Section 16.1: Custom configuration

Create a `YAML` file in the `config/` directory, for example: `config/neo4j.yml`

The content of `neo4j.yml` can be something like the below (for simplicity, `default` is used for all environments):

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

in `config/application.rb`:

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

Now, your custom config is accessible like below:

```
Rails.configuration.neo4j['host']
 #=> localhost
Rails.configuration.neo4j['port']
 #=> 7474
```

More info

Rails official API document describes the `config_for` method as:

> Convenience for loading config/foo.yml for the current Rails env.

If you do not want to use a `yaml` file

You can configure your own code through the Rails configuration object with custom configuration under the `config.x` property.

Example

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.x.super_debugger = true
```

这些配置点随后可以通过配置对象访问：

```
Rails.configuration.x.payment_processing.schedule # => :daily
Rails.configuration.x.payment_processing.retries  # => 3
Rails.configuration.x.super_debugger              # => true
Rails.configuration.x.super_debugger.not_set      # => nil
```


These configuration points are then available through the configuration object:

```
Rails.configuration.x.payment_processing.schedule # => :daily
Rails.configuration.x.payment_processing.retries  # => 3
Rails.configuration.x.super_debugger              # => true
Rails.configuration.x.super_debugger.not_set      # => nil
```

第17章：I18n - 国际化

第17.1节：带参数的I18n

你可以向I18n 方法传递参数：

```
# 示例 config/locales/en.yml
en:
  page:
users: "%{users_count} 当前在线用户数"

# 在模型、控制器等处...
I18n.('page.users', users_count: 12)

# 在视图中

# ERB
<%= t('page.users', users_count: 12) %>

#SLIM
= t('page.users', users_count: 12)

# 视图中的快捷方式 - 避免重复代码！
# 仅使用点记法
# 重要：假设你有以下控制器和视图页面#users

# ERB 示例 app/views/page/users.html.erb
<%= t('.users', users_count: 12) %>
```

并得到以下输出：

```
当前有12名用户在线
```

第17.2节：翻译ActiveRecord模型属性

globalize gem是为你的ActiveRecord模型添加翻译的绝佳解决方案。你可以通过在你的Gemfile中添加以下内容来安装它：

```
gem 'globalize', '~> 5.0.0'
```

如果你使用的是Rails 5，你还需要添加activemodel-serializers-xml

```
gem 'activemodel-serializers-xml'
```

模型翻译允许你翻译模型的属性值，例如：

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
 titlepost. # => גלובאליז 2שולט
```


Chapter 17: I18n - Internationalization

Section 17.1: I18n with arguments

You can pass parameters to **I18n** `t` method:

```
# Example config/locales/en.yml
en:
  page:
    users: "%{users_count} users currently online"

# In models, controller, etc...
I18n.t('page.users', users_count: 12)

# In views

# ERB
<%= t('page.users', users_count: 12) %>

#SLIM
= t('page.users', users_count: 12)

# Shortcut in views - DRY!
# Use only the dot notation
# Important: Consider you have the following controller and view page#users

# ERB Example app/views/page/users.html.erb
<%= t('.users', users_count: 12) %>
```

And get the following output:

```
"12 users currently online"
```

Section 17.2: Translating ActiveRecord model attributes

globalize gem is a great solution to add translations to your `ActiveRecord` models. You can install it adding this to your `Gemfile`:

```
gem 'globalize', '~> 5.0.0'
```

If you're using `Rails 5` you will also need to add `activemodel-serializers-xml`

```
gem 'activemodel-serializers-xml'
```

Model translations allow you to translate your models' attribute values, for example:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => שולט 2גלובאליז!
```

在定义了需要翻译的模型属性后，必须通过迁移创建一个翻译表。globalize 提供了 create_translation_table! 和 drop_translation_table! 方法。

对于此迁移，需要使用 up 和 down，而不是 change。并且，为了成功运行此迁移，必须先在模型中定义翻译属性，如上所示。之前的 Post 模型的合适迁移如下：

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
Post.create_translation_table! title: :string, text: :text
  end

  def down
Post.drop_translation_table!
  结束
结束
```

你也可以为特定选项传递参数，例如：

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
Post.create_translation_table! title: :string,
      text: { type: :text, null: false, default: "Default text" }
  end

  def down
Post.drop_translation_table!
  结束
结束
```

如果你在需要翻译的列中已经有任何 existing data，可以通过调整迁移轻松地将其迁移到翻译表中：

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
Post.create_translation_table!({
      title: :string,
text: :text
    }, {
migrate_data: true
    })

  end

  def down
Post.drop_translation_table! migrate_data: true
  end
end
```

确保在所有数据安全迁移后，从父表中删除已翻译的列。 若要在数据迁移后自动删除父表中的翻译列，请添加选项
remove_source_columns 到迁移中：

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
Post.create_translation_table!({
      title: :string,
text: :text
```


After you defined your model attributes that need to be translated you have to create a translation table, through a migration. **globalize** provides `create_translation_table!` and `drop_translation_table!`.

For this migration you need to use `up` and `down`, and **not** `change`. Also, in order to run this migration successfully, you have to define the translated attributes in your model first, like shown above. A proper migration for the previous `Post` model is this:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

You may also pass options for specific options, like:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string,
      text: { type: :text, null: false, default: "Default text" }
  end

  def down
    Post.drop_translation_table!
  end
end
```

In case you already have any **existing data** in your needing translation columns, you can easily migrate it to the translations table, by adjusting your migration:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true
    })

  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end
```

Make sure you drop the translated columns from the parent table after all your data is safely migrated. To automatically remove the translated columns from the parent table after the data migration, add the option `remove_source_columns` to the migration:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
```

```
    }, {
migrate_data: true,
      remove_source_columns: true
    })

  end

  def down
Post.drop_translation_table! migrate_data: true
  end
end
```

您也可以向之前创建的翻译表中添加新字段：

```
class Post < ActiveRecord::Base
  # 记得也在这里添加您的属性。
translates  :title, :text, :author
end

class AddAuthorToPost < ActiveRecord::Migration
  def up
Post.add_translation_fields! author: :text
  end

  def down
remove_column  :post_translations, :author
  end
end
```

第17.3节：从HTTP请求获取语言环境

有时根据请求的IP设置应用程序的语言环境会很有用。你可以轻松地使用Geocoder来实现这一点。除了许多功能外，Geocoder还可以告诉你请求的位置。

首先，将Geocoder添加到你的Gemfile

```
# Gemfile
gem 'geocoder'
```

Geocoder为标准的Rack::Request对象添加了location和safe_location方法，因此你可以轻松地通过IP地址查找任何HTTP请求的位置。你可以在你的

ApplicationController中使用这些方法，作为一个before_action：

```
class ApplicationController < ActionController::Base
  before_action :set_locale_from_request

  def set_locale_from_request
country_code = request.location.data["country_code"] #=> "US"
    country_sym = country_code.underscore.to_sym #=> :us

    # 如果请求的区域设置可用，则使用它，否则使用 I18n 默认区域设置
    if I18n.available_locales.include? country_sym
      I18n.locale = country_sym
    end
end
```

请注意，这在开发和测试环境中不起作用，因为像0.0.0.0和localhost这样的地址是有效的互联网 IP 地址。


```
    }, {
      migrate_data: true,
      remove_source_columns: true
    })

  end

  def down
    Post.drop_translation_table! migrate_data: true
  end
end
```

You may also add new fields to a previously created translations table:

```
class Post < ActiveRecord::Base
  # Remember to add your attribute here too.
  translates :title, :text, :author
end

class AddAuthorToPost < ActiveRecord::Migration
  def up
    Post.add_translation_fields! author: :text
  end

  def down
    remove_column :post_translations, :author
  end
end
```

Section 17.3: Get locale from HTTP request

Sometimes it can be useful to set your application locale based upon the request IP. You can easily achieve this using **Geocoder**. Among the many things **Geocoder** does, it can also tell the `location` of a `request`.

First, add **Geocoder** to your `Gemfile`

```
# Gemfile
gem 'geocoder'
```

Geocoder adds `location` and `safe_location` methods to the standard `Rack::Request` object so you can easily look up the location of any HTTP request by IP address. You can use this methods in a `before_action` in your `ApplicationController`:

```
class ApplicationController < ActionController::Base
  before_action :set_locale_from_request

  def set_locale_from_request
    country_code = request.location.data["country_code"] #=> "US"
    country_sym = country_code.underscore.to_sym #=> :us

    # If request locale is available use it, otherwise use I18n default locale
    if I18n.available_locales.include? country_sym
      I18n.locale = country_sym
    end
end
```

Beware that this will not work in `development` and `test` environments, as things like `0.0.0.0` and `localhost` are valid valid Internet IP addresses.

限制和替代方案

Geocoder 非常强大且灵活，但需要配置以配合地理编码服务（参见更多详情）；许多服务对使用次数有限制。还应注意，每次请求调用外部服务可能会影响性能。

为了解决这些问题，也可以考虑：

1. 离线解决方案

使用像GeoIP这样的 gem（参见这里）允许针对本地数据文件进行查询。可能会在准确性上有所权衡，因为这些数据文件需要保持最新。

2. 使用 CloudFlare

通过CloudFlare提供服务的页面可以选择透明地进行地理编码，国家代码将被添加到头部（`HTTP_CF_IPCOUNTRY`）。更多详情可在 here 查阅。

第17.4节：复数形式

你可以让I18n帮你处理复数形式，只需使用count参数。

你需要像这样设置你的本地化文件：

```
# config/locales/en.yml
en:
  online_users:
one: "1 user is online"
    other: "%{count} users are online"
```

然后通过传递count参数给I18n.t助手，使用你刚创建的键：

```
I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.("online_users", 计数: 4)
#=> "4 个用户在线"
```

第17.5节：通过请求设置区域设置

在大多数情况下，您可能想要设置**I18n**的区域设置。可能希望为当前会话、当前用户或基于URL参数设置区域设置。这可以通过在您的某个

控制器中，或在ApplicationController中实现一个before_action来轻松实现，从而使其作用于所有控制器。

```
class ApplicationController < ActionController::Base
  before_action :set_locale

  protected

  def set_locale
    # 移除不合适/不必要的项
I18n.locale = params[:locale] ||    # 请求参数
      session[:locale] ||               # 当前会话
      (current_user.preferred_locale if user_signed_in?) ||  # 模型保存的配置
      extract_locale_from_accept_language_header ||          # 语言头 - 浏览器配置
      I18n.default_locale               # 在配置文件中设置，默认是英语
  end
```


Limitations and alternatives

Geocoder is very powerful and flexible, but needs to be configured to work with a *geocoding service* (see more details); many of which place limits on usage. It's also worth bearing in mind that calling an external service on every request could impact performance.

To address these, it can also be worth considering:

1. An offline solution

Using a gem like **GeoIP** (see here) allows lookups to happen against a local datafile. There may be a trade-off in terms of accuracy, as these datafiles need to be kept up-to-date.

2. Use CloudFlare

Pages served through CloudFlare have the option of being geocoded transparently, with the country code being added to the header (`HTTP_CF_IPCOUNTRY`). More detail can be found here.

Section 17.4: Pluralization

You can let **I18n** handle pluralization for you, just use `count` argument.

You need to set up your locale file like this:

```
# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"
```

And then use the key you just created by passing the `count` argument to `I18n.t` helper:

```
I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"
```

Section 17.5: Set locale through requests

In most cases, you may want to set **I18n** locale. One might want to set the locale for the current session, the current user, or based on a URL parameter This is easily achievable by implementing a `before_action` in one of your controllers, or in `ApplicationController` to have it in all of your controllers.

```
class ApplicationController < ActionController::Base
  before_action :set_locale

  protected

  def set_locale
    # Remove inappropriate/unnecessary ones
    I18n.locale = params[:locale] ||    # Request parameter
      session[:locale] ||               # Current session
      (current_user.preferred_locale if user_signed_in?) ||  # Model saved configuration
      extract_locale_from_accept_language_header ||          # Language header - browser config
      I18n.default_locale               # Set in your config files, english by super-default
  end
```

```
  # 从请求头中提取语言
  def extract_locale_from_accept_language_header
    if request.env['HTTP_ACCEPT_LANGUAGE']
lg = request.env['HTTP_ACCEPT_LANGUAGE'].scan(/^[a-z]{2}/).first.to_sym
      lg.in?([:en, YOUR_AVAILABLE_LANGUAGES]) ? lg : nil
    结束
  结束
```

基于URL

该locale参数可以来自如下URL

```
http://yourapplication.com/products?locale=en
```

或者

```
http://yourapplication.com/en/products
```

要实现后者，您需要编辑您的routes，添加一个scope：

```
# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end
```

通过这样做，访问 http://yourapplication.com/en/products 将会将您的语言环境设置为 :en。相反，访问http://yourapplication.com/fr/products 将会将其设置为 :fr。此外，当缺少 :locale 参数时，访问 http://yourapplication.com/products 不会出现路由错误，因为会加载默认的 I18n 语言环境。

基于会话或基于持久化

这假设用户可以点击一个按钮/语言标志来切换语言。该操作可以路由到一个控制器，该控制器将会话设置为当前语言（如果用户已登录，最终会将更改持久化到数据库）

```
class SetLanguageController < ApplicationController
  skip_before_filter :authenticate_user!
after_action :set_preferred_locale

  # 处理大量语言的通用版本
  def change_locale
I18n.locale = sanitize_language_param
    set_session_and_redirect
  end
```

> 你必须用可用语言列表定义 sanitize_language_param，并在语言不存在时处理错误。

如果语言很少，定义方式如下可能更合适：

```
def fr
I18n.locale = :fr
  set_session_and_redirect
end

def en
```



```
  # Extract language from request header
  def extract_locale_from_accept_language_header
    if request.env['HTTP_ACCEPT_LANGUAGE']
      lg = request.env['HTTP_ACCEPT_LANGUAGE'].scan(/^[a-z]{2}/).first.to_sym
      lg.in?([:en, YOUR_AVAILABLE_LANGUAGES]) ? lg : nil
    end
  end
```

URL-based

The `locale` param could come from an URL like this

```
http://yourapplication.com/products?locale=en
```

Or

```
http://yourapplication.com/en/products
```

To achieve the latter, you need to edit your `routes`, adding a `scope`:

```
# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end
```

By doing this, visiting `http://yourapplication.com/en/products` will set your locale to `:en`. Instead, visiting `http://yourapplication.com/fr/products` will set it to `:fr`. Furthermore, you won't get a routing error when missing the `:locale` param, as visiting `http://yourapplication.com/products` will load the default **I18n** locale.

Session-based or persistence-based

This assumes the user can click on a button/language flag to change the language. The action can route to a controller that sets the session to the current language (and eventually persist the changes to a database if the user is connected)

```
class SetLanguageController < ApplicationController
  skip_before_filter :authenticate_user!
  after_action :set_preferred_locale

  # Generic version to handle a large list of languages
  def change_locale
    I18n.locale = sanitize_language_param
    set_session_and_redirect
  end
```

> You have to define sanitize_language_param with your list of available languages, and eventually handle errors in case the language doesn't exist.

If you have very few languages, it may be worth defining them like this instead:

```
def fr
  I18n.locale = :fr
  set_session_and_redirect
end

def en
```

```
I18n.locale = :en
  set_session_and_redirect
end

private

  def set_session_and_redirect
    session[:locale] = I18n.locale
    redirect_to :back
  end

  def set_preferred_locale
    if user_signed_in?
current_user.preferred_locale = I18n.locale.to_s
      current_user.save if current_user.changed?
    结束
  结束
end
```

注意：别忘了为你的change_language动作添加一些路由

默认语言环境

请记住，你需要设置应用程序的默认语言环境。你可以通过以下方式设置config/application.rb:

```
config.i18n.default_locale = :de
```

或者通过在config/initializers文件夹中创建一个初始化器：

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

第17.6节：在HTML标签和符号中使用I18n

```
# config/locales/en.yml
en：
stackoverflow：
header：
title_html: "使用 <strong>I18n</strong> 处理标签和符号"
```

注意在名称 title 后添加了额外的 _html。

在视图中，

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

第17.7节：在视图中使用I18n

假设你有以下YAML语言环境文件：

```
# config/locales/en.yml
en：
  header：
title: "我的标题"
```



```
  I18n.locale = :en
  set_session_and_redirect
end

private

  def set_session_and_redirect
    session[:locale] = I18n.locale
    redirect_to :back
  end

  def set_preferred_locale
    if user_signed_in?
      current_user.preferred_locale = I18n.locale.to_s
      current_user.save if current_user.changed?
    end
  end
end
```

Note: don't forget to add some routes to your change_language actions

Default Locale

Remember that you need to set your application default locale. You can do it by either setting it in config/application.rb:

```
config.i18n.default_locale = :de
```

or by creating an initializer in the config/initializers folder:

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

Section 17.6: Use I18n with HTML Tags and Symbols

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags &amp;amp; Symbols"
```

Note the addition of extra _html after the name title.

And in Views,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

Section 17.7: Use I18n in views

Assuming you have this YAML locale file:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

如果你想显示标题字符串，可以这样做

```
# 在ERB文件中
<%= t('header.title') %>

# 在SLIM文件中
= t('header.title')
```


and you want to display your title string, you can do this

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

第18章：在Rails中使用GoogleMaps

第18.1节：将谷歌地图JavaScript标签添加到布局头部

为了使 google maps 能够与 turbolinks 正常工作，请将 javascript 标签直接添加到布局头部，而不是包含在视图中。

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

最好在辅助方法中定义 google_maps_api_script_tag。

```
# app/helpers/google_maps_helper.rb
module GoogleMapsHelper
  def google_maps_api_script_tag
    javascript_include_tag google_maps_api_source
  end

  def google_maps_api_source
    "https://maps.googleapis.com/maps/api/js?key=#{google_maps_api_key}"
  end

  def google_maps_api_key
Rails.application.secrets.google_maps_api_key
  结束
结束
```

您可以在Google API 控制台注册您的应用并获取您的 API 密钥。Google 提供了一个简短的指南，介绍如何为 Google 地图 JavaScript API 请求 API 密钥。

API 密钥存储在secrets.yml文件中：

```
# config/secrets.yml
development:
google_maps_api_key: '...'
  # ...
production:
google_maps_api_key: '...'
  # ...
```

别忘了将config/secrets.yml添加到您的.gitignore文件中，并确保不要将 API 密钥提交到代码仓库。

第18.2节：对模型进行地理编码

假设您的用户和/或群组有个人资料，您想在谷歌地图上显示地址个人资料字段。

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # 属性：
  # 标签，例如“工作地址”
```


Chapter 18: Using GoogleMaps with Rails

Section 18.1: Add the google maps javascript tag to the layout header

In order to have google maps work properly with turbolinks, add the javascript tag directly to the layout header rather than including it in a view.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

The `google_maps_api_script_tag` is best defined in a helper.

```
# app/helpers/google_maps_helper.rb
module GoogleMapsHelper
  def google_maps_api_script_tag
    javascript_include_tag google_maps_api_source
  end

  def google_maps_api_source
    "https://maps.googleapis.com/maps/api/js?key=#{google_maps_api_key}"
  end

  def google_maps_api_key
    Rails.application.secrets.google_maps_api_key
  end
end
```

You can register your application with google and get your api key in the google api console. Google has a short guide how to request an api key for the google maps javascript api.

The api key is stored in the `secrets.yml`file:

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

Don't forget to add `config/secrets.yml` to your `.gitignore` file and makre sure you don't commit the api key to the repository.

Section 18.2: Geocode the model

Suppose, your users and/or groups have profiles and you want to display address profile fields on a google map.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # Attributes:
  # label, e.g. "Work address"
```

```
  # 值，例如 "Willy-Brandt-Straße 110557 柏林"
end
```

对地址进行地理编码（即提供经度和纬度）的一个好方法是使用geocoder gem。

将 geocoder 添加到您的Gemfile中并运行bundle进行安装。

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

添加latitude和longitude数据库列以保存位置。这比每次需要位置时都查询地理编码服务更高效。速度更快，且不会那么快达到查询限制。

```
➜ bin/rails generate migration add_latitude_and_longitude_to_profile_fields
    latitude:float longitude:float
➜ bin/rails db:migrate  # Rails 5，或者：
➜ rake db:migrate       # Rails 3，4
```

将地理编码机制添加到你的模型中。在此示例中，地址字符串存储在value属性中。
配置地理编码以在记录发生更改且存在值时执行：

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
geocoded_by :value
after_validation :geocode, if: ->(address_field){
    address_field.value.present? and address_field.value_changed?
  }

end
```

默认情况下，geocoder 使用 Google 作为查询服务。它具有许多有趣的功能，如距离计算或邻近搜索。更多信息，请查看geocoder README。

第18.3节：在个人资料视图中显示 Google 地图上的地址

在个人资料视图中，以列表形式显示用户或组的个人资料字段，以及 Google 地图上的地址字段。

```
- # app/views/profiles/show.html.haml
%h1 联系信息
.profile_fields
= render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

适当的 @profile_fields 和 @address_fields 在控制器中设置：

```
# app/controllers/profiles_controller.rb
class ProfilesController < ApplicationController
  def show
    # ...
    @profile_fields = @user_or_group.profile_fields
    @address_fields = @profile_fields.where(type: 'ProfileFields::Address')
  end
end
```



```
  # value, e.g. "Willy-Brandt-Straße 1\n10557 Berlin"
end
```

A great way to geocode the addresses, i.e. provide `longitude` and `latitude` is the geocoder gem.

Add geocoder to your `Gemfile` and run `bundle` to install it.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Add database columns for `latitude` and `longitude` in order to save the location in the database. This is more efficient than querying the geocoding service every time you need the location. It's faster and you're not hitting the query limit so quickly.

```
➜ bin/rails generate migration add_latitude_and_longitude_to_profile_fields \
    latitude:float longitude:float
➜ bin/rails db:migrate  # Rails 5, or:
➜ rake db:migrate       # Rails 3, 4
```

Add the geocoding mechanism to your model. In this example, the address string is stored in the `value` attribute. Configure the geocoding to perform when the record has changed, and only whan a value is present:

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  geocoded_by :value
  after_validation :geocode, if: ->(address_field){
    address_field.value.present? and address_field.value_changed?
  }

end
```

By default, geocoder uses google as lookup service. It has lots of interesting features like distance calculations or proximity search. Fore more information, have a look at the geocoder README.

Section 18.3: Show addresses on a google map in the profile view

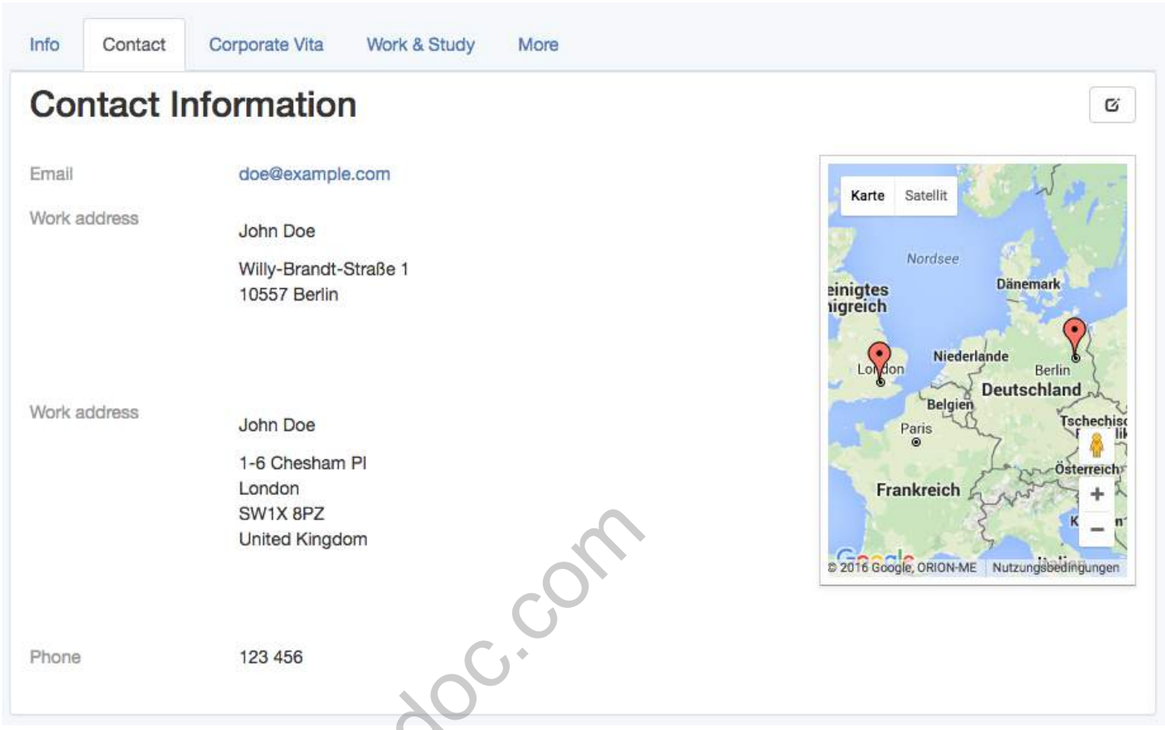
On the profile view, show the profile fields of a user or group in a list as well as the address fields on a google map.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

The appropriate `@profile_fields` and `@address_fields` are set in the controller:

```
# app/controllers/profiles_controller.rb
class ProfilesController < ApplicationController
  def show
    # ...
    @profile_fields = @user_or_group.profile_fields
    @address_fields = @profile_fields.where(type: 'ProfileFields::Address')
  end
end
```

使用 JavaScript 初始化地图，放置标记，设置缩放和其他地图设置。



第18.4节：使用 JavaScript 在地图上设置标记

假设有一个 `.google_map` div，它将成为地图，并且其有地址字段作为 markers 以 `data` 属性显示。

例如：

```
<!-- http://localhost:3000/profiles/123 -->
<div class="google_map" data-address-fields="[ {label: '
工作地址', value: '威利-布兰特大街 110557 柏林', position: {lng: ..., lat: ...}},

 ...
]"></div>
```

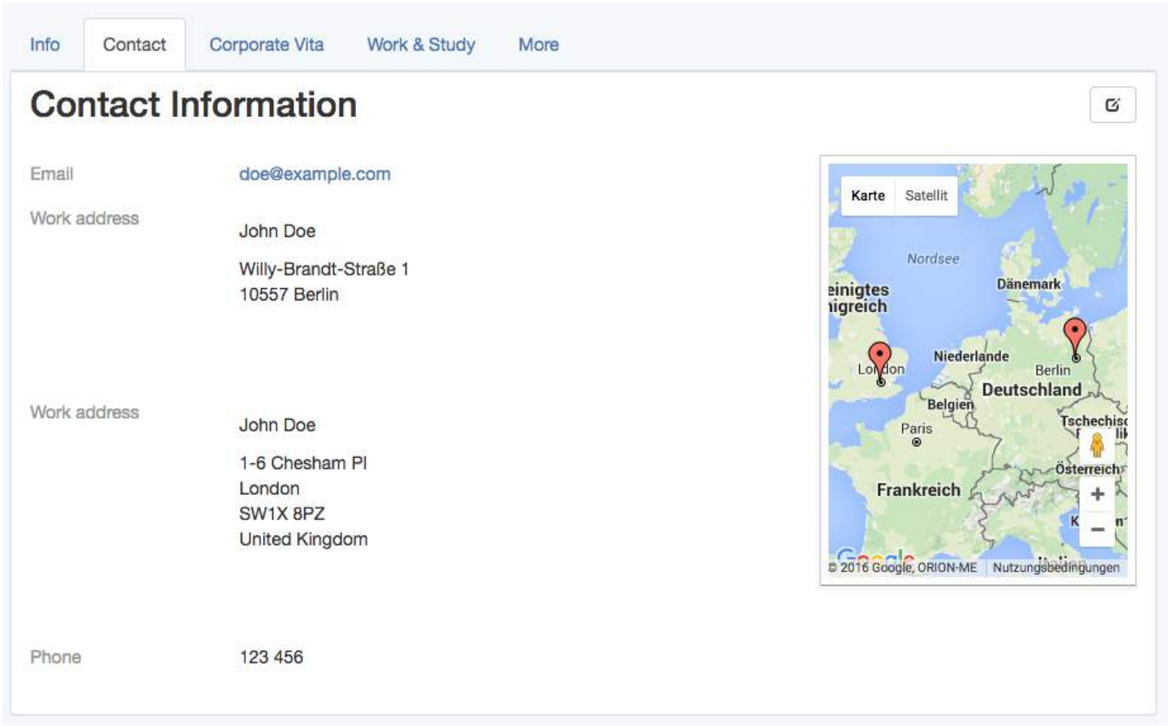
为了在不手动管理 turbolinks 事件的情况下使用 $(document).ready 事件和 turbolinks，使用 jquery.turbolinks gem。

如果你想稍后对地图执行其他操作，例如过滤或信息窗口，使用 coffee script 类 来管理地图会更方便。

```
# app/assets/javascripts/google_maps.js.coffee
window.App = {} unless App?
class App.GoogleMap
  constructor: (map_div)->
    # TODO: 初始化地图
    # TODO: 设置标记
```

当使用多个默认带命名空间的 coffee script 文件时，定义一个全局 `App` 命名空间会很方便，所有 coffee script 文件共享该命名空间。

Initialize the map, place the markers, set the zoom and other map settings with javascript.



Section 18.4: Set the markers on the map with javascript

Suppose, there is a `.google_map` div, which will become the map, and which has the address fields to show as markers as `data` attribute.

For example:

```
<!-- http://localhost:3000/profiles/123 -->
<div class="google_map" data-address-fields="[
 {label: 'Work address', value: 'Willy-Brandt-Straße 1\n10557 Berlin',
 position: {lng: ..., lat: ...}},
 ...
]"></div>
```

To make use of the `$(document).ready` event with turbolinks without managing the turbolinks events by hand, use the jquery.turbolinks gem.

If you want to perform some other operations with the map, later, for example filtering or info windows, it's convenient to have the map managed by a coffee script class.

```
# app/assets/javascripts/google_maps.js.coffee
window.App = {} unless App?
class App.GoogleMap
  constructor: (map_div)->
    # TODO: initialize the map
    # TODO: set the markers
```

When using several coffee script files, which are namespaced by default, it's convenient to define a global `App` namespace, which is shared by all coffee script files.

然后，循环遍历（可能有多个）.google_map div，并为每个div创建一个App.GoogleMap类的实例。

```
# app/assets/javascripts/google_maps.js.coffee
# ...
$(document).ready ->
  App.google_maps = []
  $('.google_map').each ->
    map_div = $(this)
    map = new App.GoogleMap map_div
    App.google_maps.push map
```

第18.5节：使用CoffeeScript类初始化地图

提供了一个App.GoogleMap CoffeeScript类，可以这样初始化谷歌地图：

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  map_div: {}
map: {}

constructor: (map_div)->
    @map_div = map_div
    @init_map()
    @reference_the_map_as_data_attribute

  # 例如，稍后通过DOM访问GoogleMap对象或地图对象本身

  #
  #     $('.google_map').data('GoogleMap')
  #
  # 将引用存储为 map_div 的数据属性。
  #
reference_the_map_as_data_attribute: ->
    @map_div.data 'GoogleMap', this
    @map_div.data 'map', @map

init_map: ->
    @map = new google.maps.Map(@dom_element, @map_configuration) if google?

  # `@map_div` 是 jQuery 对象，但 Google 地图需要
  # 真实的 DOM 元素。
  #
dom_element: ->
    @map_div.get(0)

  map_configuration: -> {
    scrollWheel: true
  }
```

要了解更多可能的 map_configuration 选项，请查看 Google 的 MapOptions 文档以及他们的 添加控件元素指南。

作为参考，类 google.maps.Map 在这里有详细的文档说明。

第18.6节：使用CoffeeScript初始化地图标记

Then, loop through (possibly several) .google_map divs and create one instance of the App.GoogleMap class for each of them.

```
# app/assets/javascripts/google_maps.js.coffee
# ...
$(document).ready ->
  App.google_maps = []
  $('.google_map').each ->
    map_div = $(this)
    map = new App.GoogleMap map_div
    App.google_maps.push map
```

Section 18.5: Initialize the map using a coffee script class

Provided an App.GoogleMap coffee script class, the google map can be initialized like this:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  map_div: {}
  map: {}

  constructor: (map_div)->
    @map_div = map_div
    @init_map()
    @reference_the_map_as_data_attribute

  # To access the GoogleMap object or the map object itself
  # later via the DOM, for example
  #
  #     $('.google_map').data('GoogleMap')
  #
  # store references as data attribute of the map_div.
  #
  reference_the_map_as_data_attribute: ->
    @map_div.data 'GoogleMap', this
    @map_div.data 'map', @map

  init_map: ->
    @map = new google.maps.Map(@dom_element, @map_configuration) if google?

  # `@map_div` is the jquery object. But google maps needs
  # the real DOM element.
  #
  dom_element: ->
    @map_div.get(0)

  map_configuration: -> {
    scrollWheel: true
  }
```

To learn more about the possible map_configuration options, have a look at google's MapOptions documentation and their guide to adding control elements.

For reference, the class google.maps.Mapis extensively documented here.

Section 18.6: Initialize the map markers using a coffee script

类

提供了一个App.GoogleMapcoffee_script类，标记信息存储在.google_mapdiv的data-address-fields属性中，地图标记可以这样初始化：

```
# app/assets/javascripts/google_maps.js.coffee
# ...
类 App.GoogleMap
  # ...
标记: []

  构造函数: (map_div)->
    # ...
    @init_markers()

address_fields: ->
    @map_div.data('address-fields')

init_markers: ->
    self = this  # 当`this`被重新定义时，用`self`引用实例。
    self.markers = []
    for address_field in self.address_fields()
      marker = new google.maps.Marker {
        map: self.map,
position: {
lng: address_field.longitude,
          lat: address_field.latitude
        },
        # # 或者，如果 `position` 在 `ProfileFields::Address#as_json` 中定义：
        # position: address_field.position,
title: address_field.value
      }
      self.markers.push marker
```

想了解更多关于标记选项的信息，请查看谷歌的 MarkerOptions 文档 以及他们的 标记指南。

第18.7节：使用coffeescript类自动缩放地图

提供了一个App.GoogleMapcoffee_script类，google.maps.Map存储为@map，google.maps.Marker存储为@markers，地图可以自动缩放，即调整到所有标记都可见，如下所示：地图如下：

```
# app/assets/javascripts/google_maps.js.coffee
# ...
类 App.GoogleMap
  # ...
边界: {}

构造函数: (map_div)->
    # ...
    @auto_zoom()

自动缩放: ->
    @init_bounds()
    # TODO: 也许，在这里调整缩放以设置最大或
    # 最小缩放级别。

初始化边界: ->
```


class

Provided an `App.GoogleMap` coffee script class and the marker information being stored in the `data-address-fields` attribute of the `.google_map` div, the map markers can be initialized on the map like this:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  markers: []

  constructor: (map_div)->
    # ...
    @init_markers()

  address_fields: ->
    @map_div.data('address-fields')

  init_markers: ->
    self = this  # to reference the instance as `self` when `this` is redefined.
    self.markers = []
    for address_field in self.address_fields()
      marker = new google.maps.Marker {
        map: self.map,
        position: {
          lng: address_field.longitude,
          lat: address_field.latitude
        },
        # # or, if `position` is defined in `ProfileFields::Address#as_json`:
        # position: address_field.position,
        title: address_field.value
      }
      self.markers.push marker
```

To learn more about marker options, have a look at google's MarkerOptions documentation and their guide to markers.

Section 18.7: Auto-zoom a map using a coffee script class

Provided an `App.GoogleMap` coffee script class with the `google.maps.Map` stored as `@map` and the `google.maps.Marker`s stored as `@markers`, the map can be auto-zoomed, i.e. adjusted that all markers are visible, like this: on the map like this:

```
# app/assets/javascripts/google_maps.js.coffee
# ...
class App.GoogleMap
  # ...
  bounds: {}

  constructor: (map_div)->
    # ...
    @auto_zoom()

  auto_zoom: ->
    @init_bounds()
    # TODO: Maybe, adjust the zoom to have a maximum or
    # minimum zoom level, here.

  init_bounds: ->
```

```
    @bounds = new google.maps.LatLngBounds()
    for marker in @markers
      @bounds.extend marker.position
    @map.fitBounds @bounds
```

想了解更多关于边界的信息，请查看谷歌的 LatLngBounds 文档。

第18.8节：将模型属性以json形式暴露

为了在谷歌地图上以标记形式显示地址配置文件字段，地址字段对象需要作为json对象传递给javascript。

常规数据库属性

当在ApplicationRecord对象上调用to_json时，数据库属性会自动被暴露。

给定一个带有`label`、value、longitude和latitude属性的ProfileFields::Address模型，`address_field.as_json`会生成一个`Hash`类型的表示，例如，

```
address_field.as_json  # =>
  {label: "工作地址", value: "Willy-Brandt-Straße 110557 柏林", longitude: ..., latitud
e: ...}
```

该表示通过to_json转换为json字符串：

```
address_field.to_json  # =>
  "{\"label\":\"工作地址\",\"value\":\"Willy-Brandt-Straße 1\\n    10557 柏林\",\"lon
gitude\":...,\"latitude\":...}"
```

这很有用，因为它允许在后续的JavaScript中使用label和value，例如用于显示地图标记的工具提示。

其他属性

可以通过重写as_json方法来暴露其他虚拟属性。

例如，要暴露一个title属性，可以将其包含在合并的as_json哈希中：

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  # 例如："约翰·多伊，工作地址"
  def title
    "#{self.parent.name}，#{self.label}"
  end

  def as_json
    super.merge {
      title: self.title
    }
  结束
结束
```

上述示例使用了super来调用原始的as_json方法，该方法返回对象的原始属性哈希，并将其与所需的位置哈希合并。


```
    @bounds = new google.maps.LatLngBounds()
    for marker in @markers
      @bounds.extend marker.position
    @map.fitBounds @bounds
```

To learn more about bounds, have a look at google's LatLngBounds documentation.

Section 18.8: Exposing the model properties as json

To display address profile fields as markers on a google map, the address field objects need to be passed as json objects to javascript.

Regular database attributes

When calling `to_json` on an `ApplicationRecord` object, the database attributes are automatically exposed.

Given a `ProfileFields::Address` model with `label`, `value`, `longitude` and `latitude` attributes, `address_field.as_json` results in a `Hash`, e.g. representation,

```
address_field.as_json  # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
    longitude: ..., latitude: ...}
```

which is converted to a json string by `to_json`:

```
address_field.to_json  # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n
    10557 Berlin\",\"longitude\":...,\"latitude\":...}"
```

This is useful because it allows to use `label` and `value` later in javascript, for example to show tool tips for the map markers.

Other attributes

Other virtual attributes can be exposed by overriding the `as_json` method.

For example, to expose a `title` attribute, include it in the merged `as_json` hash:

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  # For example: "John Doe, Work address"
  def title
    "#{self.parent.name}, #{self.label}"
  end

  def as_json
    super.merge {
      title: self.title
    }
  end
end
```

The above example uses `super` to call the original `as_json` method, which returns the original attribute hash of the object, and merges it with the required position hash.

要理解 as_json 和 to_json 之间的区别，请查看 jjulian 的 这篇博客文章。

位置

要渲染标记，谷歌地图 API 默认要求一个 position 哈希，其中经度和纬度分别存储为 lng 和 lat。

这个 position 哈希可以在 JavaScript 中创建，稍后创建，或者在定义地址字段的 JSON 表示时创建：

要将此 position 作为地址字段的 JSON 属性提供，只需在模型上重写 as_json 方法。

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  def as_json
    super.merge {
      # ...
position: {
        lng: self.longitude,
        lat: self.latitude
      }
    }
  end
end
```


To understand the difference between `as_json` and `to_json`, have a look at this blog post by jjulian.

Position

To render markers, the google maps api, by default, requires a `position` hash which has longitude and latitude stored as `lng` and `lat` respectively.

This position hash can be created in javascript, later, or here when defining the json representation of the address field:

To provide this `position` as json attribute of the address field, just override the `as_json` method on the model.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # ...

  def as_json
    super.merge {
      # ...
      position: {
        lng: self.longitude,
        lat: self.latitude
      }
    }
  end
end
```

第19章：文件上传

第19.1节：使用 Carrierwave 进行单文件上传

在 Rails 中开始使用文件上传非常简单，首先你需要选择一个用于管理上传的插件。最常见的是Carrierwave和Paperclip。两者在功能上相似，并且有丰富的文档支持

让我们来看一个使用Carrierwave上传简单头像图片的示例

在`bundle install` Carrierwave之后，在控制台输入

```
$ rails generate uploader ProfileUploader
```

这将在/app/uploaders/profile_uploader.rb创建一个配置文件

在这里你可以设置存储方式（例如本地或云端），应用图像处理扩展（例如通过MiniMagick生成缩略图），并设置服务器端的扩展名白名单

接下来，创建一个新的迁移，为user_pic设置字符串类型，并在user.rb模型中挂载上传器。

```
mount_uploader :user_pic, ProfileUploader
```

接下来，显示一个上传头像的表单（可能是用户的编辑视图）

```
<% form_for @user, html: { multipart: true } do |f| %>
    <%= f.file_field :user_pic, accept: 'image/png, image/jpg' %>
    <%= f.submit "update profile pic", class: "btn" %>
<% end %>
```

确保包含 { multipart: true }，以便表单可以处理上传。Accept是可选的，用于设置客户端的扩展名白名单。

要显示头像，只需执行

```
<%= image_tag @user.user_pic.url %>
```

第19.2节：嵌套模型 - 多文件上传

如果你想创建多个上传，首先你可能想做的是创建新模型并设置关联

假设你想为产品模型添加多张图片。创建一个新模型，并使其`belongs_to`你的父模型

```
rails g model ProductPhoto

#product.rb
has_many :product_photos, dependent: :destroy
accepts_nested_attributes_for :product_photos

#product_photo.rb
belongs_to :product
mount_uploader :image_url, ProductPhotoUploader # 确保包含上传器（Carrierwave示例）
```


Chapter 19: File Uploads

Section 19.1: Single file upload using Carrierwave

Start using File Uploads in Rails is quite simple, first thing you have to do is to choice plugin for managing uploads. The most common onces are **Carrierwave** and **Paperclip**. Both are similar in functionality and rich in documentation on

Let's have an look on example with simple avatar upload image with Carrierwave

After `bundle install` Carrierwave, type in console

```
$ rails generate uploader ProfileUploader
```

This will create an config file located at */app/uploaders/profile_uploader.rb*

Here you can set up storage (i.e local or cloud), apply extensions for image manipulations (i.e. genertings thumbs via MiniMagick) and set server-side extension white list

Next, create new migration with string tipe for user_pic and mount uploader for it in *user.rb* model.

```
mount_uploader :user_pic, ProfileUploader
```

Next, display an form to upload avatar (may be an edit view for the user)

```
<% form_for @user, html: { multipart: true } do |f| %>
    <%= f.file_field :user_pic, accept: 'image/png, image/jpg' %>
    <%= f.submit "update profile pic", class: "btn" %>
<% end %>
```

Make sure to include { multipart: true } in order form can process uploads. Accept is an optional to set client-side extension white-list.

To display an avatar, simply do

```
<%= image_tag @user.user_pic.url %>
```

Section 19.2: Nested model - multiple uploads

If you want to create multiple uploads, first thing you might want to do is create new model and set up relations

Let's say you want an multiple images for the Product model. Create an new model and make it `belongs_to` your parent model

```
rails g model ProductPhoto

#product.rb
has_many :product_photos, dependent: :destroy
accepts_nested_attributes_for :product_photos

#product_photo.rb
belongs_to :product
mount_uploader :image_url, ProductPhotoUploader # make sure to include uploader (Carrierwave example)
```

accepts_nested_attributes_for 是必须的，因为它允许我们创建嵌套表单，这样我们可以上传新文件，修改产品名称并从单个表单设置价格

接下来，在视图中创建表单（编辑/创建）

```
    <%= form_for @product, html: { multipart: true } do |product|%>

        <%= product.text_field :price # 只是普通类型的字段 %>

        <%= product.fields_for :product_photos do |photo| # 嵌套字段 %>
            <%= photo.file_field :image, :multiple =true, name: "product_photos[image_url][]" %>
        <% end %>
        <%= p.submit "Update", class: "btn" %>
    <% end %>
```

控制器没什么特别的，如果你不想新建一个，只需在你的产品控制器中创建一个新的动作

```
  # 创建一个动作
  def upload_file
printer = Product.find_by_id(params[:id])
    @product_photo = printer.prodcut_photos.create(photo_params)
  end

  # 强参数
  private
    def photo_params
params.require(:product_photos).permit(:image)
    end
```

在视图中显示所有图片

```
    <% @product.product_photos.each do |i| %>
        <%= image_tag i.image.url, class: 'img-rounded' %>
    <% end %>
```


accepts_nested_attributes_for is must, because it allow us to create nested form, so we can upload new file, change product name and set price from an single form

Next, create form in a view (edit/create)

```
    <%= form_for @product, html: { multipart: true } do |product|%>

        <%= product.text_field :price # just normal type of field %>

        <%= product.fields_for :product_photos do |photo| # nested fields %>
            <%= photo.file_field :image, :multiple => true, name: "product_photos[image_url][]" %>
        <% end %>
        <%= p.submit "Update", class: "btn" %>
    <% end %>
```

Controller is nothing special, if you don't want to create an new one, just make an new one inside your product controller

```
# create an action
def upload_file
  printer = Product.find_by_id(params[:id])
  @product_photo = printer.prodcut_photos.create(photo_params)
end

# strong params
private
  def photo_params
    params.require(:product_photos).permit(:image)
  end
```

Display all images in a view

```
    <% @product.product_photos.each do |i| %>
        <%= image_tag i.image.url, class: 'img-rounded' %>
    <% end %>
```

第20章：缓存

第20.1节：俄罗斯套娃缓存

您可能想要将缓存片段嵌套在其他缓存片段内。这称为俄罗斯套娃缓存。

俄罗斯套娃缓存的优点是，如果单个产品被更新，重新生成外层片段时，所有其他内层片段都可以被重用。

如前一节所述，如果缓存文件直接依赖的记录的updated_at值发生变化，缓存文件将过期。然而，这不会使该片段所嵌套的任何缓存过期。

例如，考虑以下视图：

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

该视图又渲染了以下视图：

```
<% cache game do %>
  <%= render game %>
<% end %>
```

如果游戏的任何属性被更改，updated_at 值将被设置为当前时间，从而使缓存失效。

但是，因为产品对象的updated_at不会被更改，所以该缓存不会失效，您的
应用程序将提供过期的数据。为了解决这个问题，我们使用touch方法将模型关联起来：

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

第20.2节：SQL缓存

查询缓存是Rails的一个功能，它缓存每个查询返回的结果集。如果Rails在同一请求中再次遇到相同的查询，它将使用缓存的结果集，而不是再次对数据库执行查询。

例如：

```
class ProductsController < ApplicationController

  def index
    # 执行查找查询
    @products = Product.all

    ...

    # 再次执行相同的查询
    @products = Product.all
  end
```


Chapter 20: Caching

Section 20.1: Russian Doll Caching

You may want to nest cached fragments inside other cached fragments. This is called `Russian doll caching`.

The advantage of `Russian doll caching` is that if a single product is updated, all the other inner fragments can be reused when regenerating the outer fragment.

As explained in the previous section, a cached file will expire if the value of `updated_at` changes for a record on which the cached file directly depends. However, this will not expire any cache the fragment is nested within.

For example, take the following view:

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

Which in turn renders this view:

```
<% cache game do %>
  <%= render game %>
<% end %>
```

If any attribute of game is changed, the `updated_at` value will be set to the current time, thereby expiring the cache.

However, because `updated_at` will not be changed for the product object, that cache will not be expired and your app will serve stale data. To fix this, we tie the models together with the touch method:

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

Section 20.2: SQL Caching

Query caching is a `Rails` feature that caches the result set returned by each query. If `Rails` encounters the same query again for that request, it will use the cached result set as opposed to running the query against the database again.

For example:

```
class ProductsController < ApplicationController

  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查询时，实际上不会访问数据库。第一次查询结果返回后，会存储在查询缓存（内存中），第二次则从内存中提取。

但是，需要注意的是，查询缓存是在一个动作开始时创建，在该动作结束时销毁，因此只在该动作期间存在。如果想以更持久的方式存储查询结果，可以使用低级缓存。

第20.3节：动作缓存

与页面缓存类似，动作缓存缓存整个页面。不同之处在于请求会经过Rails堆栈，因此在缓存被提供之前会先运行前置过滤器。该功能已从Rails中提取为 actionpack-action_caching gem。

一个常见的例子是缓存需要身份验证的动作：

```
class SecretIngredientsController < ApplicationController
  before_action :authenticate_user!, only: :index, :show
  caches_action :index

  def index
    @secret_ingredients = Recipe.find(params[:recipe_id]).secret_ingredients
  end
end
```

选项包括 :expires_in，自定义 :cache_path（用于需要不同缓存的多路由动作）以及 :if/ :unless来控制动作何时缓存。

```
class RecipesController < ApplicationController
  before_action :authenticate_user!, except: :show
  caches_page :show
caches_action :archive, expires_in: 1.day
  caches_action :index, unless: { request.format.json? }
end
```

当布局包含动态内容时，通过传递layout:false只缓存操作内容。

第20.4节：片段缓存

Rails.cache，由ActiveSupport提供，可用于跨请求缓存任何可序列化的Ruby对象。

要从缓存中获取给定键的值，使用cache.read：

```
Rails.cache.read('city')
# => nil
```

使用cache.write将值写入缓存：

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

或者，使用cache.fetch从缓存读取值，如果没有值，可以选择写入默认值：

```
Rails.cache.fetch('user') do
```



```
end
```

The second time the same query is run against the database, it's not actually going to hit the database. The first time the result is returned from the query it is stored in the query cache (in memory) and the second time it's pulled from memory.

However, it's important to note that query caches are created at the start of an action and destroyed at the end of that action and thus persist only for the duration of the action. If you'd like to store query results in a more persistent fashion, you can with low level caching.

Section 20.3: Action caching

Like page caching, action caching caches the whole page. The difference is that the request hits the Rails stack so before filters are run before the cache is served. It's extracted from Rails to actionpack-action_caching gem.

A common example is caching of an action that requires authentication:

```
class SecretIngredientsController < ApplicationController
  before_action :authenticate_user!, only: :index, :show
  caches_action :index

  def index
    @secret_ingredients = Recipe.find(params[:recipe_id]).secret_ingredients
  end
end
```

Options include `:expires_in`, a custom `:cache_path` (for actions with multiple routes that should be cached differently) and `:if`/`:unless` to control when the action should be cached.

```
class RecipesController < ApplicationController
  before_action :authenticate_user!, except: :show
  caches_page :show
  caches_action :archive, expires_in: 1.day
  caches_action :index, unless: { request.format.json? }
end
```

When the layout has dynamic content, cache only the action content by passing `layout: false`.

Section 20.4: Fragment caching

`Rails.cache`, provided by ActiveSupport, can be used to cache any serializable Ruby object across requests.

To fetch a value from the cache for a given key, use `cache.read`:

```
Rails.cache.read('city')
# => nil
```

Use `cache.write` to write a value to the cache:

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

Alternatively, use `cache.fetch` to read a value from the cache and optionally write a default if there is no value:

```
Rails.cache.fetch('user') do
```

```
User.where(:is_awesome => true)
end
```

传入块的返回值将被分配到给定键下的缓存中，然后返回。

你也可以指定缓存过期时间：

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

第20.5节：页面缓存

你可以使用ActionPack page_caching gem来缓存单个页面。它将一次动态请求的结果存储为静态HTML文件，后续请求时用该静态文件替代动态请求。README包含完整的设置说明。设置完成后，在控制器中使用caches_page类方法缓存某个动作的结果：

```
class UsersController < ActionController::Base
  caches_page :index
end
```

使用expire_page通过删除存储的HTML文件来强制缓存过期：

```
class UsersController < ActionController::Base
  caches_page :index

  def index
    @users = User.all
  end

  def create
expire_page :action => :index
  end
end
```

expire_page 的语法模仿了 url_for 及其相关方法。

第20.6节：HTTP缓存

Rails 3及以上版本开箱即用支持HTTP缓存功能。它使用 Cache-Control 和 ETag 头来控制客户端或中间代理（如CDN）缓存页面的时长。

在控制器动作中，使用 expires_in 来设置该动作的缓存时长：

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

使用 expires_now 可以强制使任何访问的客户端或中间代理立即使缓存资源过期：

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```



```
  User.where(:is_awesome => true)
end
```

The return value of the passed block will be assigned to the cache under the given key, and then returned.

You can also specify a cache expiry:

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

Section 20.5: Page caching

You can use the ActionPack page_caching gem to cache individual pages. This stores the result of one dynamic request as a static HTML file, which is served in place of the dynamic request on subsequent requests. The README contains full setup instructions. Once set up, use the caches_page class method in a controller to cache the result of an action:

```
class UsersController < ActionController::Base
  caches_page :index
end
```

Use expire_page to force expiration of the cache by deleting the stored HTML file:

```
class UsersController < ActionController::Base
  caches_page :index

  def index
    @users = User.all
  end

  def create
    expire_page :action => :index
  end
end
```

The syntax of expire_page mimics that of url_for and friends.

Section 20.6: HTTP caching

Rails >= 3 comes with HTTP caching abilities out of the box. This uses the Cache-Control and ETag headers to control how long a client or intermediary (such as a CDN) can cache a page.

In a controller action, use expires_in to set the length of caching for that action:

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

Use expires_now to force immediate expiration of a cached resource on any visiting client or intermediary:

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```

第21章：ActionController

Action Controller 是MVC中的C。路由确定使用哪个控制器处理请求后，控制器负责解析请求并生成输出。

控制器将接收请求，从模型中获取或保存数据，并使用视图来创建输出。控制器可以被视为模型和视图之间的中间人。它使模型数据可用于视图，以便向用户显示，并将用户数据保存或更新到模型中。

第21.1节：基本REST控制器

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.all
  end

  def show

  end

  def new
    @post = Post.new
  end

  def edit

  end

  def create
    @post = Post.new(post_params)

respond_to do |format|
      if @post.save
        format.html { redirect_to @post, notice: 'Post was successfully created.' }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def update
respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to @post.company, notice: '帖子已成功更新。' }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def destroy
    @post.destroy
    respond_to do |format|
```


Chapter 21: ActionController

Action Controller is the C in MVC. After the router has determined which controller to use for a request, the controller is responsible for making sense of the request and producing the output.

The controller will receive the request, fetch or save data from a model and use a view to create output. A controller can be thought of as a middleman between models and views. It makes the model data available to the view so it can display to the user, and it saves or updates user data to the model.

Section 21.1: Basic REST Controller

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.all
  end

  def show

  end

  def new
    @post = Post.new
  end

  def edit

  end

  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
        format.html { redirect_to @post, notice: 'Post was successfully created.' }
        format.json { render :show, status: :created, location: @post }
      else
        format.html { render :new }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to @post.company, notice: 'Post was successfully updated.' }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  def destroy
    @post.destroy
    respond_to do |format|
```

```
      format.html { redirect_to posts_url, notice: '帖子已成功删除。' }
      format.json { head :no_content }
    end
  end

  private
    def set_post
      @post = Post.find(params[:id])
    end

    def post_params
params.require(:post).permit(:title, :body, :author)
    end
end
```

第21.2节：过滤器

过滤器是“前置”、“后置”或“环绕”控制器动作执行的方法。它们是继承的，因此如果你在ApplicationController中设置了任何过滤器，它们将在应用程序接收到的每个请求中运行。

前置过滤器

前置过滤器在控制器动作执行之前运行，可以中止请求（和/或重定向）。一个常见的用途是验证用户是否已登录：

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
redirect_to some_path unless user_signed_in?
  结束
结束
```

在请求到达控制器的动作之前，会先运行过滤器。它本身可以返回一个响应，并完全绕过该动作。

before过滤器的其他常见用途是验证用户的身份认证，以便在授予他们访问处理请求的指定动作之前进行验证。我也见过它们用于从数据库加载资源、检查资源权限，或在其他情况下管理重定向。

After过滤器

After过滤器类似于“before”过滤器，但由于它们在动作执行后运行，因此可以访问即将发送的响应对象。简而言之，after过滤器在动作完成后运行。它可以修改响应。大多数情况下，如果在after过滤器中完成某些操作，也可以在动作本身中完成，但如果有一些逻辑需要在执行一组动作后运行，那么after过滤器是一个很好的位置。

通常，我见过after和around过滤器用于日志记录。

Around过滤器

Around过滤器可能在动作执行前后都有逻辑。它会在必要的位置让出执行权给动作。注意，它不一定需要让出执行权，也可以像before过滤器一样不让出执行权而运行。

Around过滤器负责通过让出执行权来运行其关联的动作，类似于Rack中间件的工作方式。


```
      format.html { redirect_to posts_url, notice: 'Post was successfully destroyed.' }
      format.json { head :no_content }
    end
  end

  private
    def set_post
      @post = Post.find(params[:id])
    end

    def post_params
      params.require(:post).permit(:title, :body, :author)
    end
end
```

Section 21.2: Filters

Filters are methods that are run "before", "after" or "around" a controller action. They are inherited, so if you set any in your `ApplicationController` they will be run for every request your application receives.

Before Filter

Before filters are executed before the controller action and can halt the request (and/or redirect). A common use is to verify if a user is logged in:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

Before filters are run on requests before the request gets to the controller's action. It can return a response itself and completely bypass the action.

Other common uses of before filters is validating a user's authentication before granting them access to the action designated to handle their request. I've also seen them used to load a resource from the database, check permissions on a resource, or manage redirects under other circumstances.

After Filter

After filters are similar to "before" ones, but as they get executed after the action run they have access the response object that's about to be sent. So in short after filters are run after the action completes. It can modify the response. Most of the time if something is done in an after filter, it can be done in the action itself, but if there is some logic to be run after running any of a set of actions, then an after filter is a good place to do it.

Generally, I've seen after and around filters used for logging.

Around Filter

Around filters may have logic before and after the action being run. It simply yields to the action in whatever place is necessary. Note that it doesn't need to yield to the action and may run without doing so like a before filter.

Around filters are responsible for running their associated actions by yielding, similar to how Rack middlewares work.

Around回调包裹动作的执行。你可以用两种不同的风格编写around回调。第一种是回调是一段代码块。该代码在动作执行前调用。如果回调代码调用yield，则动作被执行。当动作完成后，回调代码继续执行。因此，yield之前的代码类似于before动作回调，yield之后的代码类似于after动作回调。如果回调代码从未调用yield，则动作不会执行——这与before动作回调返回false相同。

以下是around过滤器的示例：

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
logger.debug "捕获异常！#{e.message}"
    end
  end
```

这将捕获任何操作的异常并将消息记录到日志中。您可以使用around过滤器进行异常处理、设置和拆卸，以及其他各种情况。

Only 和 Except

所有过滤器都可以应用于特定操作，使用 :only 和 :except：

```
class ProductsController < ApplicationController
  before_action :set_product, only: [:show, :edit, :update]

  # ... 控制器操作

  # 将过滤器定义为控制器的私有方法
  private

  def set_product
    @product = Product.find(params[:id])
  end
end
```

跳过过滤器

所有过滤器（包括继承的过滤器）也可以针对某些特定操作跳过：

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
redirect_to some_path unless user_signed_in?
  end
end

class HomeController < ApplicationController
  skip_before_action :authenticate_user!, only: [:index]

  def index
  end
end
```


Around callbacks wrap the execution of actions. You can write an around callback in two different styles. In the first, the callback is a single chunk of code. That code is called before the action is executed. If the callback code invokes yield, the action is executed. When the action completes, the callback code continues executing. Thus, the code before the yield is like a before action callback and the code after the yield is the after action callback. If the callback code never invokes yield. the action is not run-this is the same as having a before action callback return false.

Here's an example of the around filter:

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
      logger.debug "Caught exception! #{e.message}"
    end
  end
```

This will catch exception of any action and put the message in your log. You can use around filters for exception handling, setup and teardown, and a myriad of other cases.

Only and Except

All filters can be applied to specific actions, using `:only` and `:except`:

```
class ProductsController < ApplicationController
  before_action :set_product, only: [:show, :edit, :update]

  # ... controller actions

  # Define your filters as controller private methods
  private

  def set_product
    @product = Product.find(params[:id])
  end
end
```

Skipping Filter

All filters (inherited ones too) can also be skipped for some specific actions:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end

class HomeController < ApplicationController
  skip_before_action :authenticate_user!, only: [:index]

  def index
  end
end
```

由于过滤器是继承的，也可以在命名空间的“父”控制器中定义过滤器。比如说你有一个admin命名空间，当然你只希望管理员用户能够访问它。你可以这样做：

```
# config/routes.rb
namespace :admin do
  resources :products
end

# app/controllers/admin_controller.rb
class AdminController < ApplicationController
  before_action :authenticate_admin_user!

  private

  def authenticate_admin_user!
redirect_to root_path unless current_user.admin?
  结束
结束

# app/controllers/admin/products_controller.rb
class Admin::ProductsController < AdminController
  # This controller will inherit :authenticate_admin_user! filter
end
```

请注意，在Rails 4.x中你可以同时使用before_filter和before_action，但before_filter在Rails 5.0.0中已被弃用，并将在5.1版本中移除。

第21.3节：生成控制器

Rails提供了许多生成器，当然也包括控制器生成器。

你可以通过在应用程序文件夹中运行以下命令来生成一个新的控制器

```
rails generate controller NAME [action action] [options]
```

注意：您也可以使用 `rails g` 别名来调用 `rails generate`

例如，要为 `Product` 模型生成一个包含 *`#index`* 和 *`#show`* 操作的控制器，您可以运行

```
rails generate controller products index show
```

这将在app/controllers/products_controller.rb中创建控制器，包含你指定的两个动作

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

它还会在app/views/目录下创建一个products文件夹，包含控制器动作对应的两个模板（即index.html.erb和show.html.erb，注意扩展名可能根据你的模板引擎有所不同，比如如果你使用的是slim，生成器会创建index.html.slim和show.html.slim）

此外，如果你指定了任何动作，它们也会被添加到你的routes文件中

As they're inherited, filters can also be defined in a `namespace` "parent" controller. Say for example that you have an `admin` namespace, and you of course want only admin users to be able to access it. You could do something like this:

```
# config/routes.rb
namespace :admin do
  resources :products
end

# app/controllers/admin_controller.rb
class AdminController < ApplicationController
  before_action :authenticate_admin_user!

  private

  def authenticate_admin_user!
    redirect_to root_path unless current_user.admin?
  end
end

# app/controllers/admin/products_controller.rb
class Admin::ProductsController < AdminController
  # This controller will inherit :authenticate_admin_user! filter
end
```

Beware that in **Rails 4.x** you could use `before_filter` along with `before_action`, but `before_filter` is currently deprecated in **Rails 5.0.0** and will be removed in **5.1**.

Section 21.3: Generating a controller

Rails provides a lot of generators, for controllers too of course.

You can generate a new controller by running this command in your app folder

```
rails generate controller NAME [action action] [options]
```

Note: You can also use `rails g` alias to invoke `rails generate`

For example, to generate a controller for a `Product` model, with *`#index`* and *`#show`* actions you would run

```
rails generate controller products index show
```

This will create the controller in `app/controllers/products_controller.rb`, with both the actions you specified

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

It will also create a `products` folder inside `app/views/`, containing the two templates for your controller's actions (i.e. `index.html.erb` and `show.html.erb`, *note that the extension may vary according to your template engine, so if you're using `slim`, for example, generator will create `index.html.slim` and `show.html.slim`*)

Furthermore, if you specified any actions they will also be added to your `routes` file

```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails会为你创建一个辅助文件，位于app/helpers/products_helper.rb，同时还会创建资源文件，分别在app/assets/javascripts/products.js和app/assets/stylesheets/products.css。至于视图，生成器会根据你Gemfile中的配置调整行为：例如，如果你的应用使用了Coffeescript和Sass，控制器生成器会改为生成products.coffee和products.sass文件。

最后但同样重要的是，Rails还会为你的控制器、辅助和视图生成测试文件。

如果你不想生成这些文件，可以告诉Rails跳过它们，只需在任何选项前加上

```
--no-
```

或者 --skip，像这样：

```
rails generate controller products index show --no-assets --no-helper
```

生成器将跳过 `assets` 和 `helper`

如果你需要为特定的 **`namespace`** 创建控制器，将其添加在 `NAME` 前面：

```
rails generate controller admin/products
```

这将在 `app/controllers/admin/products_controller.rb` 中创建你的控制器。

Rails 也可以为你生成一个完整的 RESTful 控制器：

```
rails generate scaffold_controller MODEL_NAME # 从 Rails 4 开始可用
rails generate scaffold_controller Product
```

第21.4节：使用redirect_to捕获 ActiveRecord::RecordNotFound 异常

你可以通过重定向来捕获 RecordNotFound 异常，而不是显示错误页面：

```
class ApplicationController < ActionController::Base

  # 你的其他内容

rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: I18n.t("errors.record_not_found")
  end
end
```

第21.5节：为异常显示错误页面

如果你想向用户显示有意义的错误信息，而不是简单的“抱歉，出了点问题”，Rails 提供了一个很好的工具来实现这个目的。

打开文件 app/controllers/application_controller.rb，你应该会看到类似如下内容：

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
```



```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails creates a helper file for you, in `app/helpers/products_helper.rb`, and also the assets files in `app/assets/javascripts/products.js` and `app/assets/stylesheets/products.css`. As for views, the generator changes this behaviour according to what's specified in your `Gemfile`: i.e., if you're using `Coffeescript` and `Sass` in your application, the controller generator will instead generator `products.coffee` and `products.sass`.

At last, but not least, Rails also generates test files for your controller, your helper and your views.

If you don't want any of these to be created for you can tell Rails to skip them, just prepend any option with

```
--no-
```

or `--skip`, like this:

```
rails generate controller products index show --no-assets --no-helper
```

And the generator will skip both `assets` and `helper`

If you need to create a controller for a specific **`namespace`** add it in front of `NAME`:

```
rails generate controller admin/products
```

This will create your controller inside `app/controllers/admin/products_controller.rb`

Rails can also generate a complete RESTful controller for you:

```
rails generate scaffold_controller MODEL_NAME # available from Rails 4
rails generate scaffold_controller Product
```

Section 21.4: Rescuing ActiveRecord::RecordNotFound with redirect_to

You can rescue a RecordNotFound exception with a redirect instead of showing an error page:

```
class ApplicationController < ActionController::Base

  # your other stuff

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: I18n.t("errors.record_not_found")
  end
end
```

Section 21.5: Display error pages for exceptions

If you want to display to your users meaningful errors instead of simple "sorry, something went wrong", Rails has a nice utility for the purpose.

Open the file `app/controllers/application_controller.rb` and you should find something like this:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
```

```
end
```

我们现在可以添加一个 rescue_from 来处理特定的错误：

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
rescue_from ActiveRecord::RecordNotFound, with: :record_not_found

  private

  def record_not_found
render html: "Record <strong>not found</strong>", status: 404
  结束
结束
```

建议不要捕获 Exception 或 StandardError，否则 Rails 将无法在发生错误时显示有用的错误页面。

第21.6节：输出JSON而非HTML

```
class UsersController < ApplicationController
  def index
hashmap_or_array = [{ name: "foo", email: "foo@example.org" }]

    respond_to do |format|
      format.html { render html: "Hello World" }
      format.json { render json: hashmap_or_array }
    end
  结束
结束
```

此外，你还需要以下路由：

```
resources :users, only: [:index]
```

这将对 /users 的请求以两种不同方式响应：

- 如果你访问 `/users` 或 `/users.html`，它将显示内容为 `Hello World`
- 的HTML页面；如果你访问 `/users.json`，它将显示包含以下内容的JSON对象：

```
[
  {
    "name": "foo",
    "email": "foo@example.org"
  }
]
```

你可以 omit format.html { render inline: "Hello World" }，如果你想确保你的路由只响应 JSON 请求。

第21.7节：控制器（基础）

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html { render html: "Hello World" }
    end
```



```
end
```

We can now add a `rescue_from` to recover from specific errors:

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  rescue_from ActiveRecord::RecordNotFound, with: :record_not_found

  private

  def record_not_found
    render html: "Record <strong>not found</strong>", status: 404
  end
end
```

It's recommended not to rescue from **`Exception`** or **`StandardError`** otherwise Rails won't be able to display helpful pages in case of errors.

Section 21.6: Output JSON instead of HTML

```
class UsersController < ApplicationController
  def index
    hashmap_or_array = [{ name: "foo", email: "foo@example.org" }]

    respond_to do |format|
      format.html { render html: "Hello World" }
      format.json { render json: hashmap_or_array }
    end
  end
end
```

In addition you will need the route:

```
resources :users, only: [:index]
```

This will respond in two different ways to requests on `/users`:

- If you visit `/users` or `/users.html`, it will show an html page with the content `Hello World`
- If you visit `/users.json`, it will display a JSON object containing:

```
[
  {
    "name": "foo",
    "email": "foo@example.org"
  }
]
```

You can **omit** `format.html { render inline: "Hello World" }` if you want to make sure that your route will answer only to JSON requests.

Section 21.7: Controllers (Basic)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render html: "Hello World" }
    end
```

```
  结束
结束
```

这是一个基础控制器，附加了以下路由（在 routes.rb 中）：

```
resources :users, only: [:index]
```

当你访问 URL `/users` 时，会在网页中显示 Hello World 消息

第21.8节：参数

控制器可以访问 HTTP 参数（你可能知道它们在 URL 中表现为 ?name=foo，但 Ruby on Rails 也处理不同格式！），并根据参数输出不同的响应。无法区分 GET 和 POST 参数，但无论如何你都不应该这样做。

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html do
        if params[:name] == "john"
          render html: "Hello John"
        else
渲染 HTML: "Hello someone"
        结束
      结束
    end
  end
end
```

像往常一样，我们的路由：

```
resources :users, only: [:index]
```

访问 URL `/users?name=john`，输出将是 `Hello John`，访问 `/users?name=whatever`，输出将是 `Hello someone`

第21.9节：过滤参数（基础）

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html do
渲染 HTML: "Hello #{ user_params[:name] } user_params[:sentence]"
      结束
    结束
  end

  private

  def user_params
    if params[:name] == "john"
      params.permit(:name, :sentence)
    else
params.permit(:name)
    end
  结束
结束
```



```
  end
end
```

This is a basic controller, with the addition of the following route (in routes.rb):

```
resources :users, only: [:index]
```

Will display the `Hello World` message in a webpage when you access the URL `/users`

Section 21.8: Parameters

Controllers have access to HTTP parameters (you might know them as `?name=foo` in URLs, but Ruby on Rails handle different formats too!) and output different responses based on them. There isn't a way to distinguish between GET and POST parameters, but you shouldn't do that in any case.

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        if params[:name] == "john"
          render html: "Hello John"
        else
          render html: "Hello someone"
        end
      end
    end
  end
end
```

As usual our route:

```
resources :users, only: [:index]
```

Access the URL `/users?name=john` and the output will be `Hello John`, access `/users?name=whatever` and the output will be `Hello someone`

Section 21.9: Filtering parameters (Basic)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        render html: "Hello #{ user_params[:name] } user_params[:sentence]"
      end
    end
  end

  private

  def user_params
    if params[:name] == "john"
      params.permit(:name, :sentence)
    else
      params.permit(:name)
    end
  end
end
```

你可以允许（或拒绝）某些参数，这样只有你想要的参数会通过，避免出现用户设置不应更改的选项等意外情况。

访问 /users?name=john&sentence=developer 会显示 Hello john developer，但访问 /users?name=smith&sentence=spy 只会显示 Hello smith，因为 :sentence 只有在以 john 身份访问时才被允许

第21.10节：重定向

假设路由：

```
resources :users, only: [:index]
```

你可以使用以下方式重定向到不同的URL：

```
class UsersController
  def index
redirect_to "http://stackoverflow.com/"
  结束
结束
```

你可以使用以下方式返回用户之前访问的页面：

```
redirect_to :back
```

注意在Rails 5中，返回重定向的语法有所不同：

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

该语法会尝试重定向到之前的页面，如果不可能（浏览器阻止了HTTP_REFERRER头部），则会重定向到`:fallback_location`

第21.11节：使用视图

假设路由：

```
resources :users, only: [:index]
```

控制器如下：

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html { render }
    end
  结束
结束
```

视图 app/users/index.html.erb 将被渲染。如果视图是：

```
Hello <strong>World</strong>
```

输出将是一个包含文本："Hello **World**"的网页

如果你想渲染不同的视图，可以使用：

You can allow (or reject) some params so that only what you want will *pass through* and you won't have bad surprises like user setting options not meant to be changed.

Visiting `/users?name=john&sentence=developer` will display `Hello john developer`, however visiting `/users?name=smith&sentence=spy` will display `Hello smith` only, because `:sentence` is only allowed when you access as `john`

Section 21.10: Redirecting

Assuming the route:

```
resources :users, only: [:index]
```

You can redirect to a different URL using:

```
class UsersController
  def index
    redirect_to "http://stackoverflow.com/"
  end
end
```

You can go back to the previous page the user visited using:

```
redirect_to :back
```

Note that in *Rails 5* the syntax for redirecting back is different:

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

Which will try to redirect to the previous page and in case not possible (the browser is blocking the HTTP_REFERRER header), it will redirect to `:fallback_location`

Section 21.11: Using Views

Assuming the route:

```
resources :users, only: [:index]
```

And the controller:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

The view `app/users/index.html.erb` will be rendered. If the view is:

```
Hello <strong>World</strong>
```

The output will be a webpage with the text: "Hello **World**"

If you want to render a different view, you can use:

```
render "pages/home"
```

则会使用文件 app/views/pages/home.html.erb 代替。

你可以使用控制器实例变量向视图传递变量：

```
class UsersController < ApplicationController
  def index
    @name = "john"

respond_to do |format|
      format.html { render }
    end
  结束
结束
```

在文件app/views/users/index.html.erb中，你可以使用@name：

```
Hello <strong><%= @name %></strong>
```

输出将是："Hello john"

关于渲染语法的重要说明，你可以完全省略render语法，Rails 会假设你省略了它。所以：

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html { render }
    end
  结束
结束
```

可以改写为：

```
class UsersController < ApplicationController
  def index
respond_to do |format|
      format.html
    结束
  结束
end
```

Rails 足够智能，能够判断必须渲染文件app/views/users/index.html.erb。


```
render "pages/home"
```

And the file app/views/pages/home.html.erb will be used instead.

You can pass variables to views using controller instance variables:

```
class UsersController < ApplicationController
  def index
    @name = "john"

    respond_to do |format|
      format.html { render }
    end
  end
end
```

And in the file app/views/users/index.html.erb you can use @name:

```
Hello <strong><%= @name %></strong>
```

And the output will be: "Hello **john**"

An important note around the render syntax, you can omit the `render` syntax entirely, Rails assumes that if you omit it. So:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

Can be written instead as:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html
    end
  end
end
```

Rails is smart enough to figure out that it must render the file app/views/users/index.html.erb.

第22章：配置

第22.1节：Rails 通用配置

以下配置选项应在Rails::Railtie对象上调用

- config.after_initialize：接受一个块，该块将在 Rails 初始化应用程序后运行。
- **config.asset_host**：设置资源的主机。当使用*内容分发网络（Content Delivery Network）*时，这非常有用。这是config.action_controller.asset_host的简写。
- config.autoload_once_paths：此选项接受一个路径数组，Rails 会在这些路径下自动加载常量。默认值为空数组。
- config.autoload_paths：此选项接受一个路径数组，Rails 会在这些路径下自动加载常量。默认包含app目录下的所有子目录。
- config.cache_classes：决定是否在每次请求时重新加载类和模块。在开发模式下，默认值为false；在生产和测试模式下，默认值为trueconfig.action_view.cache_template_loading：决定是否在每次请求
- 时重新加载模板。默认值与config.cache_classes设置相同。

- config.beginning_of_week：设置默认的一周开始日。需要一个有效的星期符号（:monday）。

- **config.cache_store**：选择使用哪种缓存存储。选项包括`:file_store`、`:memory_store`、`mem_cache_store`或`null_store`。
- **config.colorize_logging**：控制日志信息是否带颜色显示。
- **config.eager_load**：预加载所有注册的内容。
- **config.encoding**：指定应用的编码。默认值为`UTF-8`
- **config.log_level**：设置 Rails 日志记录器的详细级别。所有环境下默认值为`:debug`。
- **config.middleware**：用于配置应用的中间件。
- **config.time_zone**：设置应用的默认时区。

第22.2节：配置资源

以下配置选项可用于配置资产

- **config.assets.enabled**：确定是否启用资产管道。默认值为true
- **config.assets.raise_runtime_errors：启用运行时错误检查。对开发模式很有用**
- **config.assets.compress**：允许压缩资产。在生产模式下，默认值为true
- config.assets.js_compressor：指定使用哪种JS压缩器。选项包括:closure、:uglifier和:yui

- config.assets.paths：指定搜索资产的路径。
- **config.assets.precompile**：允许你选择在运行 rake `assets:precompile`时预编译的额外资产
- config.assets.digest：此选项允许在资产名称中使用MD-5指纹。开发模式下默认值为true

- config.assets.compile：切换生产模式下的实时Sprockets编译

第22.3节：配置生成器

Rails允许你配置运行`rails generate`命令时使用的生成器。此方法，`config.generators`接受一个代码块

```
config.generators do |g|
g.orm :active_record
```


Chapter 22: Configuration

Section 22.1: Rails General Configuration

The following configuration options should be called on a `Rails::Railtie` object

- **config.after_initialize**: Takes a block which will be run after rails has initialized the application.
- **config.asset_host**: This sets the host for the assets. This is useful when using a *Content Delivery Network*. This is shorthand for `config.action_controller.asset_host`
- **config.autoload_once_paths**: This option accepts an array of paths where Rails autoloads constants. The default value is an empty array
- **config.autoload_paths**: This accepts an array of paths where Rails autoloads constants. It defaults to all directories under `app`
- **config.cache_classes**: Determines if classes and modules should be reloaded on each request. In development mode, this defaults to `false` and in the production and test modes it defaults to `true`
- **config.action_view.cache_template_loading**: This determines if templates should be reloaded on each request. It defaults to the `config.cache_classes` setting
- **config.beginning_of_week**: This sets the default beginning of week. It requires a valid week day symbol (`:monday`)
- **config.cache_store**: Choose which cache store to use. Options include `:file_store`, `:memory_store`, `mem_cache_store` or `null_store`.
- **config.colorize_logging**: This controls whether logging information is colorized
- **config.eager_load**: Eager-loads all registered
- **config.encoding**: Specifies the application encoding. The default value is `UTF-8`
- **config.log_level**: Sets the verbosity of the Rails Logger. It defaults to `:debug` in all environments.
- **config.middleware**: Use this to configure the application's middleware
- **config.time_zone**: This sets the application's default time zone.

Section 22.2: Configuring assets

The following configuration options can be used for configuring assets

- **config.assets.enabled**: Determines whether the asset pipeline is enabled. This defaults to true
- **config.assets.raise_runtime_errors**: This enables runtime error checking. It's useful for `development mode`
- **config.assets.compress**: Lets assets be compressed. In production mode, this defaults to true
- **config.assets.js_compressor**: Specifies which JS compressor to use. Options include `:closure`, `:uglifier` and `:yui`
- **config.assets.paths**: Specifies which paths to search for assets.
- **config.assets.precompile**: Lets you choose additional assets to be precompiled when `rake assets:precompile` is run
- **config.assets.digest**: This option allows the use of `MD-5` fingerprints in the asset names. It defaults to true in development mode
- **config.assets.compile**: Toggles live `Sprockets` compilation in production mode

Section 22.3: Configuring generators

Rails allows you to configure what generators are used when running `rails generate` commands. This method, `config.generators` takes a block

```
config.generators do |g|
  g.orm :active_record
```

```
g.test_framework :test_unit
end
```

以下是一些选项

选项	描述	默认
资源	生成脚手架时创建资源	true
force_plural	允许模型名称使用复数形式	false
helper	决定是否生成辅助方法	true
integration_tool	指定集成工具	`测试单元`
javascript_engine	配置JS引擎	`:js`
资源路由	生成资源路由	true
stylesheet_engine	配置样式表引擎	`:cs`
scaffold_stylesheet	在脚手架生成时创建CSS	true
测试框架	指定测试框架	`Minitest`
模板引擎	配置模板引擎	`:erb`

第22.4节：Rails中的环境

Rails的配置文件位于`config/environments/`目录下。默认情况下，Rails有3个环境，`development`、`production`和`test`。通过编辑每个文件，您只是在编辑该环境的配置。

Rails还有一个配置文件位于config/application.rb。这个是通用配置文件，因为这里定义的任何设置都会被各环境中指定的配置覆盖。

您可以在Rails.application.configure do块中添加或修改配置选项，配置选项以config.开头。

第22.5节：数据库配置

Rails 项目的数据库配置位于文件`config/database.yml`中。如果你使用`rails new`命令创建项目且未指定要使用的数据库引擎，则 Rails 默认使用`sqlite`作为数据库。一个典型的带有默认配置的`database.yml`文件大致如下所示。

```
# SQLite 版本 3.x
#   安装 sqlite3 gem
#
#   确保在你的 Gemfile 中定义了 SQLite 3 gem
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
timeout: 5000

development:
  <<: *default
database: db/development.sqlite3

# 警告：定义为“test”的数据库将在运行“rake”时被清除并
# 从你的开发数据库重新生成。
# 不要将此数据库设置为与开发或生产数据库相同。
test:
  <<: *default
database: db/test.sqlite3
```



```
  g.test_framework :test_unit
end
```

Here are some of the options

Option	Description	Default
assets	Creates assets when generating scaffold	true
force_plural	Allows pluralized model names	false
helper	Determines whether to generate helpers	true
integration_tool	Specify integration tool	`test_unit`
javascript_engine	Configures JS engine	`:js`
resource_route	Generates resource route	true
stylesheet_engine	Configures stylesheet engine	`:cs`
scaffold_stylesheet	Creates CSS upon scaffolding	true
test_framework	Specify Test Framework	`Minitest`
template_engine	Configures template engine	`:erb`

Section 22.4: Environments in Rails

Configuration files for rails can be found in `config/environments/`. By default rails has 3 environments, `development`, `production` and `test`. By editing each file you are editing the configuration for that environment only.

Rails also has a configuration file in `config/application.rb`. This is a common configuration file as any settings defined here are overwritten by the config specified in each environment.

You add or modify configuration options within the `Rails.application.configure do` block and configuration options start with `config.`

Section 22.5: Database Configuration

Database configuration of a rails project lies in a file `config/database.yml`. If you create a project using `rails new` command and don't specify a database engine to be used then rails uses `sqlite` as the default database. A typical `database.yml` file with default configuration will look similar to following.

```
# SQLite version 3.x
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3
```

```
production:
  <<: *default
database: db/production.sqlite3
```

如果你想在创建新项目时更改默认数据库，可以指定数据库：`rails new hello_world --database=mysql`


```
production:
  <<: *default
  database: db/production.sqlite3
```

If you want to change the default database while creating a new project you can specify database: `rails new hello_world --database=mysql`

第23章：安全常量化

第23.1节：成功的safe_constantize

`User` 是一个 `ActiveRecord` 或 `Mongoid` 类。将 `User` 替换为你项目中的任何 `Rails` 类（甚至像 **`Integer`** 或 **`Array`** 这样的）

```
my_string = "User" # 首字母大写的字符串
  # => 'User'
my_constant = my_string.safe_constantize
  # => User
my_constant.all.count
  # => 18

my_string = "Array"
  # => '数组'
my_constant = my_string.safe_constantize
  # => 数组
my_constant.new(4)
  # => [nil, nil, nil, nil]
```

第23.2节：safe_constantize失败的情况

此示例无法工作，因为传入的字符串在项目中未被识别为常量。即使你传入`"array"`，也无法工作，因为它没有大写。

```
my_string = "not_a_constant"
  # => 'not_a_constant'
my_string.safe_constantize
  # => nil

my_string = "array" #未大写！
  # => 'array'
my_string.safe_constantize
  # => nil
```


Chapter 23: Safe Constantize

Section 23.1: Successful safe_constantize

`User` is an `ActiveRecord` or `Mongoid` class. Replace `User` with any `Rails` class in your project (even something like **`Integer`** or **`Array`**)

```
my_string = "User" # Capitalized string
  # => 'User'
my_constant = my_string.safe_constantize
  # => User
my_constant.all.count
  # => 18

my_string = "Array"
  # => 'Array'
my_constant = my_string.safe_constantize
  # => Array
my_constant.new(4)
  # => [nil, nil, nil, nil]
```

Section 23.2: Unsuccessful safe_constantize

This example will not work because the string passed in isn't recognized as a constant in the project. Even if you pass in `"array"`, it won't work as it isn't capitalized.

```
my_string = "not_a_constant"
  # => 'not_a_constant'
my_string.safe_constantize
  # => nil

my_string = "array" #Not capitalized!
  # => 'array'
my_string.safe_constantize
  # => nil
```

第24章：Rails 5

第24.1节：如何在RVM上安装Ruby on Rails 5

RVM 是管理你的 Ruby 版本和设置工作环境的绝佳工具。

假设你已经安装了 RVM，要获取这些示例所需的最新 Ruby 版本，
请打开终端并运行：

```
$ rvm get stable
$ rvm install ruby --latest
```

通过运行以下命令检查你的 Ruby 版本：

```
$ ruby -v
> ruby 2.3.0p0
```

要安装 Rails 5，首先使用最新的 Ruby 版本创建一个新的 gemset，然后安装 Rails：

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

要检查你的 Rails 版本，运行：

```
$ rails -v
> Rails 5.0.0
```

第 24.2 节：创建 Ruby on Rails 5 API

要创建一个新的 Rails 5 API，请打开终端并运行以下命令：

```
rails new app_name --api
```

将创建以下文件结构：

```
   create
create  README.rdoc
   create   Rakefile
create  config.ru
create  .gitignore
   create   Gemfile
   create   app
create  app/assets/javascripts/application.js
   create   app/assets/stylesheets/application.css
   create   app/controllers/application_controller.rb
   create   app/helpers/application_helper.rb
  创建  app/views/layouts/application.html.erb
  创建  app/assets/images/.keep
   create   app/mailers/.keep
   create   app/models/.keep
   create   app/controllers/concerns/.keep
   create   app/models/concerns/.keep
  create   bin
创建 bin/bundle
创建 bin/rails
创建 bin/rake
```


Chapter 24: Rails 5

Section 24.1: How to install Ruby on Rails 5 on RVM

RVM is a great tool to manage your ruby versions and set up your working environment.

Assuming you already have RVM installed, to get the latest version of ruby, which is needed for these examples, open a terminal and run:

```
$ rvm get stable
$ rvm install ruby --latest
```

Check your ruby version by running:

```
$ ruby -v
> ruby 2.3.0p0
```

To install Rails 5, first create a new gemset using the latest ruby version and then install rails:

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

To check your rails version, run:

```
$ rails -v
> Rails 5.0.0
```

Section 24.2: Creating a Ruby on Rails 5 API

To create a new Rails 5 API, open a terminal and run the following command:

```
rails new app_name --api
```

The following file structure will be created:

```
  create
  create  README.rdoc
  create  Rakefile
  create  config.ru
  create  .gitignore
  create  Gemfile
  create  app
  create  app/assets/javascripts/application.js
  create  app/assets/stylesheets/application.css
  create  app/controllers/application_controller.rb
  create  app/helpers/application_helper.rb
  create  app/views/layouts/application.html.erb
  create  app/assets/images/.keep
  create  app/mailers/.keep
  create  app/models/.keep
  create  app/controllers/concerns/.keep
  create  app/models/concerns/.keep
  create  bin
  create  bin/bundle
  create  bin/rails
  create  bin/rake
```

```
创建 bin/setup
创建配置
  创建  config/routes.rb
  创建  config/application.rb
  创建  config/environment.rb
  创建  config/secrets.yml
  创建  config/environments
  创建  config/environments/development.rb
  创建  config/environments/production.rb
  创建  config/environments/test.rb
  创建  config/initializers
  创建  config/initializers/assets.rb
  创建  config/initializers/backtrace_silencers.rb
  创建  config/initializers/cookies_serializer.rb
  创建  config/initializers/filter_parameter_logging.rb
  创建  config/initializers/inflections.rb
   create  config/initializers/mime_types.rb
   create  config/initializers/session_store.rb
   create  config/initializers/wrap_parameters.rb
   create  config/locales
  创建  config/locales/en.yml
  创建  config/boot.rb
创建 config/database.yml
创建 db
create  db/seeds.rb
  create  lib
create  lib/tasks
   create  lib/tasks/.keep
   create  lib/assets
   create  lib/assets/.keep
  create  log
create  log/.keep
  create  public
   create  public/404.html
   create  public/422.html
   create  public/500.html
   create  public/favicon.ico
   create  public/robots.txt
   create  test/fixtures
   create  test/fixtures/.keep
   create  test/controllers
   create  test/controllers/.keep
   create  test/mailers
create  test/mailers/.keep
   create  test/models
create  test/models/.keep
   create  test/helpers
   create  test/helpers/.keep
   create  test/integration
   create  test/integration/.keep
   create  test/test_helper.rb
   create  tmp/cache
create  tmp/cache/assets
   create  vendor/assets/javascripts
   create  vendor/assets/javascripts/.keep
   create  vendor/assets/stylesheets
create  vendor/assets/stylesheets/.keep
```

该文件结构将被创建在一个名为app_name的新文件夹内。它包含启动项目所需的所有资源和代码。


```
  create  bin/setup
  create  config
  create  config/routes.rb
  create  config/application.rb
  create  config/environment.rb
  create  config/secrets.yml
  create  config/environments
  create  config/environments/development.rb
  create  config/environments/production.rb
  create  config/environments/test.rb
  create  config/initializers
  create  config/initializers/assets.rb
  create  config/initializers/backtrace_silencers.rb
  create  config/initializers/cookies_serializer.rb
  create  config/initializers/filter_parameter_logging.rb
  create  config/initializers/inflections.rb
  create  config/initializers/mime_types.rb
  create  config/initializers/session_store.rb
  create  config/initializers/wrap_parameters.rb
  create  config/locales
  create  config/locales/en.yml
  create  config/boot.rb
  create  config/database.yml
  create  db
  create  db/seeds.rb
  create  lib
  create  lib/tasks
  create  lib/tasks/.keep
  create  lib/assets
  create  lib/assets/.keep
  create  log
  create  log/.keep
  create  public
  create  public/404.html
  create  public/422.html
  create  public/500.html
  create  public/favicon.ico
  create  public/robots.txt
  create  test/fixtures
  create  test/fixtures/.keep
  create  test/controllers
  create  test/controllers/.keep
  create  test/mailers
  create  test/mailers/.keep
  create  test/models
  create  test/models/.keep
  create  test/helpers
  create  test/helpers/.keep
  create  test/integration
  create  test/integration/.keep
  create  test/test_helper.rb
  create  tmp/cache
  create  tmp/cache/assets
  create  vendor/assets/javascripts
  create  vendor/assets/javascripts/.keep
  create  vendor/assets/stylesheets
  create  vendor/assets/stylesheets/.keep
```

This file structure will be created inside a new folder called `app_name`. It contains all the assets and code needed to start your project.

进入该文件夹并安装依赖：

```
cd app_name
bundle install
```

你还应该启动你的数据库。Rails 默认使用 SQLite 作为数据库。要创建它，请运行：

```
rake db:setup
```

现在运行你的应用程序：

```
$ rails server
```

当你在浏览器中打开 http://localhost:3000 时，你崭新的（空的）API 应该正在运行！

Enter the folder and install the dependencies:

```
cd app_name
bundle install
```

You should also start your database. Rails uses SQLite as a default database. To create it, run:

```
rake db:setup
```

Now run your appplication:

```
$ rails server
```

When you open your browser at `http://localhost:3000`, your shiny new (empty) API should be running!

第25章：使用 CanCan 进行授权

CanCan 是一个简单的 Rails 授权策略，与用户角色解耦。所有权限都存储在一个位置。

第25.1节：开始使用 CanCan

CanCan 是一个流行的 Ruby on Rails 授权库，用于限制用户对特定资源的访问。最新的 gem（CanCanCan）是已停止维护项目 CanCan 的延续。

权限定义在 Ability 类中，可以在控制器、视图、助手或代码中的任何其他地方使用。

要为应用程序添加授权支持，请将 CanCanCan gem 添加到Gemfile中：

```
gem 'cancancan'
```

然后定义 Ability 类：

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
  end
end
```

然后使用 load_and_authorize_resource 来加载控制器中的授权模型并进行授权检查：

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show
    # @article 已经被加载并授权
  end
end
```

使用 authorize! 来检查授权或抛出异常

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

can? 用于检查对象是否被授权执行控制器、视图或辅助方法中的特定操作

```
<% if can? :update, @article %>
  <%= link_to "编辑", edit_article_path(@article) %>
<% end %>
```

注意： 这假设已登录用户由current_user方法提供。

第25.2节：处理大量权限

一旦权限定义的数量开始增加，处理

Chapter 25: Authorization with CanCan

CanCan is a simple authorization strategy for Rails which is decoupled from user roles. All permissions are stored in a single location.

Section 25.1: Getting started with CanCan

CanCan is a a popular authorization library for Ruby on Rails which restricts user access to specific resources. The latest gem (CanCanCan) is a continuation of the dead project CanCan.

Permissions are defined in the `Ability` class and can be used from controllers, views, helpers, or any other place in the code.

To add authorization support to an app, add the CanCanCan gem to the `Gemfile`:

```
gem 'cancancan'
```

Then define the ability class:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
  end
end
```

Then check authorization using `load_and_authorize_resource` to load authorized models into the controller:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show
    # @article is already loaded and authorized
  end
end
```

`authorize!` to check authorization or raise an exception

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

can? to check if an object is authorized against a particular action anywhere in the controllers, views, or helpers

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

Note: This assumes the signed user is provided by the `current_user` method.

Section 25.2: Handling large number of abilities

Once the number of abilities definitions start to grow in number, it becomes more and more difficult to handle the

Ability 文件变得越来越困难。

处理这些问题的第一种策略是将权限移入有意义的方法中，如下例所示：

```
class Ability
  include CanCan::Ability

  def initialize(user)
   anyone_abilities

    if user
      如果 用户。管理员？
        admin_abilities
      else
        authenticated_abilities
      end
    else
      guest_abilities
    结束
  结束

  private

  def anyone_abilities
    # 为所有人定义权限，包括登录用户和访客
  end

  def guest_abilities
    # 仅为访客定义权限
  end

  def authenticated_abilities
    # 仅为登录用户定义权限
  end

  def admin_abilities
    # 仅为管理员定义权限
  end
end
```

一旦这个类变得足够大，你可以尝试将其拆分成不同的类来处理不同的职责，像这样：

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    self.merge Abilities::Everyone.new(user)

    if user
      如果 用户。管理员？
        self.merge Abilities::Admin.new(user)
      else
        self.merge Abilities::Authenticated.new(user)
      end
    else
      self.merge Abilities::Guest.new(user)
    end
  end
```


Ability file.

The first strategy to handle these issue is to move abilities into meaningful methods, as per this example:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    anyone_abilities

    if user
      if user.admin?
        admin_abilities
      else
        authenticated_abilities
      end
    else
      guest_abilities
    end
  end

  private

  def anyone_abilities
    # define abilities for everyone, both logged users and visitors
  end

  def guest_abilities
    # define abilities for visitors only
  end

  def authenticated_abilities
    # define abilities for logged users only
  end

  def admin_abilities
    # define abilities for admins only
  end
end
```

Once this class grow large enough, you can try breaking it into different classes to handle the different responsibilities like this:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    self.merge Abilities::Everyone.new(user)

    if user
      if user.admin?
        self.merge Abilities::Admin.new(user)
      else
        self.merge Abilities::Authenticated.new(user)
      end
    else
      self.merge Abilities::Guest.new(user)
    end
  end
```

```
end
```

然后将这些类定义为：

```
# app/models/abilities/guest.rb
module Abilities
  class Guest
    include CanCan::Ability

    def initialize(user)
      # 仅限匿名访客的权限
    end
  结束
结束
```

等等，使用Abilities::Authenticated、Abilities::Admin或其他任何权限。

第25.3节：定义能力

能力在Ability类中使用can和cannot方法定义。请参考以下带注释的示例作为基本参考：

```
class Ability
  include CanCan::Ability

  def initialize(user)
    # 对于任何访客或用户
    can :read, Article

    if user
      如果 用户。管理员？
        # 管理员可以对任何模型或操作执行任何操作
can :manage, :all
      else
        # 普通用户可以阅读所有内容
can :read, :all
        # 并且只能编辑、更新和删除自己的用户
        can [:edit, :destroy], User, id: user_id
        # 但不能阅读隐藏的文章
cannot :read, Article, hidden: true
      end
    else
      # 只有未登录的访客可以访问注册页面：
can :read, :sign_up
    end
  结束
结束
```

第25.4节：快速测试权限

如果您想快速测试一个权限类是否赋予了正确的权限，可以在控制台或加载了Rails环境的其他上下文中初始化一个权限，只需传入一个用户实例进行测试：

```
test_ability = Ability.new(User.first)
test_ability.can?(:show, Post) #=> true
other_ability = Ability.new(RestrictedUser.first)
other_ability.cannot?(:show, Post) #=> true
```



```
end
```

and then define those classes as:

```
# app/models/abilities/guest.rb
module Abilities
  class Guest
    include CanCan::Ability

    def initialize(user)
      # Abilities for anonymous visitors only
    end
  end
end
```

and so on with `Abilities::Authenticated`, `Abilities::Admin` or any other else.

Section 25.3: Defining abilities

Abilities are defined in the `Ability` class using `can` and `cannot` methods. Consider the following commented example for basic reference:

```
class Ability
  include CanCan::Ability

  def initialize(user)
    # for any visitor or user
    can :read, Article

    if user
      if user.admin?
        # admins can do any action on any model or action
        can :manage, :all
      else
        # regular users can read all content
        can :read, :all
        # and edit, update and destroy their own user only
        can [:edit, :destroy], User, id: user_id
        # but cannot read hidden articles
        cannot :read, Article, hidden: true
      end
    else
      # only unlogged visitors can visit a sign_up page:
      can :read, :sign_up
    end
  end
end
```

Section 25.4: Quickly test an ability

If you'd like to quickly test if an ability class is giving the correct permissions, you can initialize an ability in the console or on another context with the rails environment loaded, just pass an user instance to test against:

```
test_ability = Ability.new(User.first)
test_ability.can?(:show, Post) #=> true
other_ability = Ability.new(RestrictedUser.first)
other_ability.cannot?(:show, Post) #=> true
```

更多信息：https://github.com/ryanb/cancan/wiki/Testing-Abilities

More information: https://github.com/ryanb/cancan/wiki/Testing-Abilities

第26章：Mongoid

第26.1节：字段

根据Mongoid文档，有16种有效的字段类型：

- 数组
- 大数
- 布尔
- 日期
- DateTime
- 浮点数
- 哈希
- 整数
- BSON::ObjectId
- BSON::二进制
- 范围
- 正则表达式
- 字符串
- 符号
- 时间
- 带时区的时间

要添加一个字段（我们称之为name，类型为String），请将以下内容添加到你的模型文件中：

```
字段:name，类型：String
```

要设置默认值，只需传入default选项：

```
字段 :名称，类型：字符串，默认值：""
```

第26.2节：安装

首先，将 Mongoid 添加到你的 Gemfile 中：

```
gem "mongoid", "~> 4.0.0"
```

然后运行 bundle install。或者直接运行：

```
$ gem install mongoid
```

安装完成后，运行生成器以创建配置文件：

```
$ rails g mongoid:config
```

这将创建文件 (myapp)/config/mongoid.yml。

第26.3节：创建模型

通过运行以下命令创建一个模型（我们称之为User）：

Chapter 26: Mongoid

Section 26.1: Fields

As per the Mongoid Documentation, there are 16 valid field types:

- Array
- BigDecimal
- Boolean
- Date
- DateTime
- Float
- Hash
- Integer
- BSON::ObjectId
- BSON::Binary
- Range
- Regexp
- String
- Symbol
- Time
- TimeWithZone

To add a field (let's call it `name` and have it be a `String`), add this to your model file:

```
field :name, type: String
```

To set a default value, just pass in the `default` option:

```
field :name, type: String, default: ""
```

Section 26.2: Installation

First, add `Mongoid` to your `Gemfile`:

```
gem "mongoid", "~> 4.0.0"
```

and then run `bundle install`. Or just run:

```
$ gem install mongoid
```

After installation, run the generator to create the config file:

```
$ rails g mongoid:config
```

which will create the file `(myapp)/config/mongoid.yml`.

Section 26.3: Creating a Model

Create a model (lets call it `User`) by running:

```
$ rails g model User
```

这将生成文件app/models/user.rb：

```
类 User
  include Mongoid::Document

end
```

这就是你创建模型所需的全部内容（虽然只有一个id字段）。与ActiveRecord不同，没有迁移文件。模型的所有数据库信息都包含在模型文件中。

生成模型时，时间戳不会自动包含在模型中。要向模型添加created_at和updated_at字段，请添加

```
include Mongoid::Timestamps
```

到模型中，放在include Mongoid::Document下面，如下所示：

```
类 User
  include Mongoid::Document
  include Mongoid::Timestamps

end
```

第26.4节：经典关联

Mongoid 允许经典的 ActiveRecord 关联：

- 一对一：has_one / belongs_to
- 一对多：has_many / belongs_to
- 多对多：has_and_belongs_to_many

要添加关联（比如 User has_many posts），你可以将以下内容添加到你的 User 模型文件中：

```
has_many :posts
```

并将以下内容添加到你的 Post 模型文件中：

```
belongs_to :user
```

这将在你的 Post 模型中添加一个 user_id 字段，给 Post 类添加一个 user 方法，并给你的 User 类添加一个 posts 方法。

第26.5节：嵌入式关联

Mongoid 允许嵌入式关联：

- 一对一：embeds_one / embedded_in
- 一对多：embeds_many / embedded_in

要添加一个关联（比如用户embeds_many地址），请将以下内容添加到您的User文件中：

```
embeds_many :addresses
```



```
$ rails g model User
```

which will generate the file app/models/user.rb:

```
class User
  include Mongoid::Document

end
```

This is all you need to have a model (albeit nothing but an id field). Unlike ActiveRecord, there is no migration files. All the database information for the model is contained in the model file.

Timestamps are not automatically included in your model when you generate it. To add created_at and updated_at to your model, add

```
include Mongoid::Timestamps
```

to your model underneath include Mongoid::Document like so:

```
class User
  include Mongoid::Document
  include Mongoid::Timestamps

end
```

Section 26.4: Classic Associations

Mongoid allows the classic ActiveRecord associations:

- One-to-one: has_one / belongs_to
- One-to-many: has_many / belongs_to
- Many-to-many: has_and_belongs_to_many

To add an association (lets say the User has_many posts), you can add this to your User model file:

```
has_many :posts
```

and this to your Post model file:

```
belongs_to :user
```

This will add a user_id field in your Post model, add a user method to your Post class, and add a posts method to your User class.

Section 26.5: Embedded Associations

Mongoid allows Embedded Associations:

- One-to-one: embeds_one / embedded_in
- One-to-many: embeds_many / embedded_in

To add an association (lets say the User embeds_many addresses), add this to your User file:

```
embeds_many :addresses
```

以及将此添加到你的 Address 模型文件中：

```
embedded_in :user
```

这将在你的 User 模型中嵌入 Address，给你的 User 类添加一个 addresses 方法。

第26.6节：数据库调用

Mongoid 尽量采用与 ActiveRecord 类似的语法。它支持这些调用（以及更多）

```
User.first #从数据库获取第一个用户

User.count #获取数据库中所有用户的数量

User.find(params[:id]) #返回 params[:id] 中找到的用户

User.where(name: "Bob") #返回一个 Mongoid::Criteria 对象，可以链式调用
                        #其他查询（如另一个 'where' 或 'any_in'）
                        #不会返回数据库中的任何对象

User.where(name: "Bob").entries #从数据库返回所有名字为 "Bob" 的对象User.where(:name.in => ['Bo

b', 'Alice']).entries #从数据库返回所有名字为 "Bob" 或 "Alice" 的对象


User.any_in(name: ["Bob", "Joe"]).first #返回第一个名字为 "Bob" 或 "Joe" 的对象
User.where(:name => 'Bob').exists? #如果存在一个或多个名字为 bob 的用户，则返回 true
```


and this to your Address model file:

```
embedded_in :user
```

This will embed Address in your User model, adding a addresses method to your User class.

Section 26.6: Database Calls

Mongoid tries to have similar syntax to ActiveRecord when it can. It supports these calls (and many more)

```
User.first #Gets first user from the database

User.count #Gets the count of all users from the database

User.find(params[:id]) #Returns the user with the id found in params[:id]

User.where(name: "Bob") #Returns a Mongoid::Criteria object that can be chained
                        #with other queries (like another 'where' or an 'any_in')
                        #Does NOT return any objects from database

User.where(name: "Bob").entries #Returns all objects with name "Bob" from database

User.where(:name.in => ['Bob', 'Alice']).entries #Returns all objects with name "Bob" or "Alice"
from database

User.any_in(name: ["Bob", "Joe"]).first #Returns the first object with name "Bob" or "Joe"
User.where(:name => 'Bob').exists? # will return true if there is one or more users with name bob
```

第27章：Gems

第27.1节：Gemfiles

首先，gemfiles 至少需要一个源，形式为 RubyGems 服务器的 URL。

通过运行bundle init生成一个带有默认 rubygems.org 源的 Gemfile。使用 https 以便您的服务器连接通过 SSL 进行验证。

```
source 'https://rubygems.org'
```

接下来，声明您需要的 gems，包括版本号。

```
gem 'rails', '4.2.6'
gem 'rack',  '>=1.1'
gem 'puma',  '~>3.0'
```

大多数版本说明符，如 >= 1.0，含义自明。说明符 ~> 有特殊含义。~> 2.0.3 等同于 >= 2.0.3 且 < 2.1。~> 2.1 等同于 >= 2.1 且 < 3.0。~> 2.2.beta 会匹配预发布版本，如 2.2.beta.12。

Git 仓库也是有效的 gem 源，只要仓库包含一个或多个有效的 gems。使用:tag、:branch或:ref指定检出内容。默认是master分支。

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

如果您想直接使用文件系统中解包的 gem，只需将 :path 选项设置为包含该 gem 文件的路径。

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

依赖项可以被分组。组可以在安装时被忽略（使用--without）或一次性全部要求（使用Bundler.require）。

```
gem 'rails_12factor'，组：:production

group :development, :test do
gem 'byebug'
gem 'web-console'，'~> 2.0'
  gem 'spring'
gem 'dotenv-rails'
end
```

你可以在Gemfile中使用ruby指定所需的Ruby版本。如果Gemfile在不同的Ruby版本上加载，Bundler会抛出带有说明的异常。

```
ruby '2.3.1'
```

第27.2节：什么是gem？

gem相当于编程语言Ruby的插件或扩展。

确切地说，甚至Rails也不过是一个gem。许多gem是基于Rails或其他gem构建的（它们依赖于所述gem），或者是独立的。

Chapter 27: Gems

Section 27.1: Gemfiles

To start, gemfiles require at least one source, in the form of the URL for a RubyGems server.

Generate a Gemfile with the default rubygems.org source by running `bundle init`. Use https so your connection to the server will be verified with SSL.

```
source 'https://rubygems.org'
```

Next, declare the gems that you need, including version numbers.

```
gem 'rails', '4.2.6'
gem 'rack',  '>=1.1'
gem 'puma',  '~>3.0'
```

Most of the version specifiers, like >= 1.0, are self-explanatory. The specifier ~> has a special meaning. ~> 2.0.3 is identical to >= 2.0.3 and < 2.1. ~> 2.1 is identical to >= 2.1 and < 3.0. ~> 2.2.beta will match prerelease versions like 2.2.beta.12.

Git repositories are also valid gem sources, as long as the repo contains one or more valid gems. Specify what to check out with `:tag`, `:branch`, or `:ref`. The default is the `master` branch.

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

If you would like to use an unpacked gem directly from the filesystem, simply set the :path option to the path containing the gem's files.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Dependencies can be placed into groups. Groups can be ignored at install-time (using `--without`) or required all at once (using `Bundler.require`).

```
gem 'rails_12factor', group: :production

group :development, :test do
  gem 'byebug'
  gem 'web-console', '~> 2.0'
  gem 'spring'
  gem 'dotenv-rails'
end
```

You can specify the required version of Ruby in the Gemfile with `ruby`. If the Gemfile is loaded on a different Ruby version, Bundler will raise an exception with an explanation.

```
ruby '2.3.1'
```

Section 27.2: What is a gem?

A gem is the equivalent to a plugin or an extension for the programming language ruby.

To be exact even rails is nothing more than a gem. A lot of gems are built on rails or other gems (they are dependent of said gem) or are standalone.

在你的 Rails 项目中
Gemfile

对于你的 Rails 项目，你有一个名为Gemfile的文件。在这里你可以添加你想包含和使用的 gems。添加后，你需要使用bundler安装这些 gem（参见 Bundler 部分）。

Gemfile.lock

完成后，你的Gemfile.lock文件将会更新，包含你新添加的 gems 及其依赖项。该文件锁定你使用的 gems，使它们使用该文件中声明的特定版本。

```
GEM
remote: https://rubygems.org/
specs:
devise (4.0.3)
bcrypt (~> 3.0)
orm_adapter (~> 0.1)
railties (>= 4.1.0, < 5.1)
responders
warden (~> 1.2.3)
```

此示例针对 gemdevise。在Gemfile.lock中声明了版本4.0.3，用于告诉在其他机器或生产服务器上安装你的项目时使用指定的版本。

开发

无论是个人、团队还是整个社区，都在维护和开发一个 gem。完成的工作通常会在修复了某些issues或添加了features后发布。

通常发布遵循语义化版本控制 2.0.0原则。

第27.3节：Bundler

管理和处理 gems 最简单的方法是使用bundler。Bundler是一个类似于bower的包管理器。

要使用 bundler，首先需要安装它。

```
gem install bundler
```

安装并运行 bundler 后，你只需将 gems 添加到你的Gemfile中，然后运行

```
bundle
```

在终端中。这会将新添加的 gems 安装到你的项目中。如果出现问题，终端会提示你。

如果你想了解更多细节，建议查看docs。

第27.4节：Gemsets（宝石集）

如果你使用RVM（Ruby版本管理器），那么为每个项目使用一个gemset（宝石集）是个好主意。一个gemset只是一个容器，你可以用它来将宝石彼此分开。为每个项目创建一个gemset可以让你更改某个项目的宝石（和宝石版本），而不会破坏你所有其他项目。每个项目只需关心自己的宝石。

In your Rails project
Gemfile

For your Rails project you have a file called `Gemfile`. In here you can add gems you want to include and use in your project. Once added you need to install the gem by using `bundler` (See Bundler section).

Gemfile.lock

Once you have done this, your `Gemfile.lock` will be updated with your newly added gems and their dependencies. This file locks your used gems so they use that specific version declared in that file.

```
GEM
remote: https://rubygems.org/
specs:
devise (4.0.3)
bcrypt (~> 3.0)
orm_adapter (~> 0.1)
railties (>= 4.1.0, < 5.1)
responders
warden (~> 1.2.3)
```

This example is for the gem `devise`. In the `Gemfile.lock` the version `4.0.3` is declared, to tell when installing your project on an other machine or on your production server which specified version to use.

Development

Either a single person, a group or a whole community works on and maintains a gem. Work done is usually released after certain `issues` have been fixed or `features` have been added.

Usually the releases follow the Semantic Versioning 2.0.0 principle.

Section 27.3: Bundler

The easiest way to handle and manage gems is by using `bundler`. Bundler is a package manager comparable to bower.

To use bundler you first need to install it.

```
gem install bundler
```

After you have bundler up and running all you need to do is add gems to your `Gemfile` and run

```
bundle
```

in your terminal. This installs your newly added gems to your project. Should an issue arise, you would get a prompt in your terminal.

If you are interested in more details, I suggest you have a look at the docs.

Section 27.4: Gemsets

If you are using `RVM(Ruby Version Manager)` then using a `gemset` for each project is a good idea. A `gemset` is just a container you can use to keep gems separate from each other. Creating a `gemset` per project allows you to change gems (and gem versions) for one project without breaking all your other projects. Each project need only worry about its own gems.

RVM（版本>= 0.1.8）为每个Ruby解释器提供一个@global宝石集。你安装到某个Ruby的@global宝石集中的宝石，可以被你为该Ruby创建的所有其他宝石集使用。这是让所有项目共享特定Ruby解释器安装的相同已安装宝石的好方法。

创建宝石集

假设你已经安装了ruby-2.3.1，并且你已经使用以下命令选择了它：

```
rvm use ruby-2.3.1
```

现在为该Ruby版本创建宝石集：

```
rvm gemset create new_gemset
```

其中new_gemset是宝石集的名称。要查看某个Ruby版本可用的宝石集列表：

```
rvm gemset list
```

列出所有 Ruby 版本的 gem：

```
rvm gemset list_all
```

使用列表中的某个 gemset（假设new_gemset是我想使用的 gemset）：

```
rvm gemset use new_gemset
```

如果想切换到其他 Ruby 版本，也可以指定 Ruby 版本和 gemset：

```
rvm use ruby-2.1.1@new_gemset
```

为特定 Ruby 版本指定默认的 gemset：

```
rvm use 2.1.1@new_gemset --default
```

要移除 gemset 中所有已安装的 gem，可以通过以下命令清空：

```
rvm gemset empty new_gemset
```

要将一个 Ruby 的 gemset 复制到另一个 Ruby，可以这样做：

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

要删除一个 gemset：

```
rvm gemset delete new_gemset
```

查看当前的 gemset 名称：

```
rvm gemset name
```

在全局 gemset 中安装一个 gem：

```
rvm @global do gem install ...
```


RVM provides (>= 0.1.8) a `@global gemset` per ruby interpreter. Gems you install to the `@global gemset` for a given ruby are available to all other gemsets you create in association with that ruby. This is a good way to allow all of your projects to share the same installed gem for a specific ruby interpreter installation.

Creating gemsets

Suppose you already have `ruby-2.3.1` installed and you have selected it using this command:

```
rvm use ruby-2.3.1
```

Now to create gemset for this ruby version:

```
rvm gemset create new_gemset
```

where the `new_gemset` is the name of gemset. To see the list of available gemsets for a ruby version:

```
rvm gemset list
```

to list the gems of all ruby versions:

```
rvm gemset list_all
```

to use a gemset from the list (suppose `new_gemset` is the gemset I want to use):

```
rvm gemset use new_gemset
```

you can also specify the ruby version with the gemset if you want to shift to some other ruby version:

```
rvm use ruby-2.1.1@new_gemset
```

to specify a default gemset for a particular ruby version:

```
rvm use 2.1.1@new_gemset --default
```

to remove all the installed gems from a gemset you can empty it by:

```
rvm gemset empty new_gemset
```

to copy a gemset from one ruby to another you can do it by:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

to delete a gemset:

```
rvm gemset delete new_gemset
```

to see the current gemset name:

```
rvm gemset name
```

to install a gem in the global gemset:

```
rvm @global do gem install ...
```

在安装 Ruby 时初始化 Gemsets

当你安装一个新的 Ruby 时，RVM 不仅会创建两个 gemset（默认的空 gemset 和全局 gemset），它还会使用一组用户可编辑的文件来确定要安装哪些 gem。

在 ~/.rvm/gemsets 目录下工作时，rvm 会根据安装的 ruby 版本字符串，使用基于树状层级的方式搜索 global.gems 和 default.gems。以 ree-1.8.7-p2010.02 为例，rvm 会检查（并从中导入）以下文件：

```
~/.rvm/gemsets/ree/1.8.7/p2010.02/global.gems
~/.rvm/gemsets/ree/1.8.7/p2010.02/default.gems
~/.rvm/gemsets/ree/1.8.7/global.gems
~/.rvm/gemsets/ree/1.8.7/default.gems
~/.rvm/gemsets/ree/global.gems
~/.rvm/gemsets/ree/default.gems
~/.rvm/gemsets/global.gems
~/.rvm/gemsets/default.gems
```

例如，如果你编辑了 ~/.rvm/gemsets/global.gems 文件，添加了以下两行：

```
bundler
awesome_print
```

每次安装新的 ruby 时，这两个 gem 都会被安装到你的全局 gemset 中。 default.gems 和 global.gems 文件通常会在 rvm 更新时被覆盖。

Initializing Gemsets during Ruby Installs

When you install a new ruby, RVM not only creates two gemsets (the default, empty gemset and the global gemset), it also uses a set of user-editable files to determine which gems to install.

Working in `~/.rvm/gemsets`, rvm searchs for `global.gems` and `default.gems` using a tree-hierachy based on the ruby string being installed. Using the example of `ree-1.8.7-p2010.02`, rvm will check (and import from) the following files:

```
~/.rvm/gemsets/ree/1.8.7/p2010.02/global.gems
~/.rvm/gemsets/ree/1.8.7/p2010.02/default.gems
~/.rvm/gemsets/ree/1.8.7/global.gems
~/.rvm/gemsets/ree/1.8.7/default.gems
~/.rvm/gemsets/ree/global.gems
~/.rvm/gemsets/ree/default.gems
~/.rvm/gemsets/global.gems
~/.rvm/gemsets/default.gems
```

For example, if you edited `~/.rvm/gemsets/global.gems` by adding these two lines:

```
bundler
awesome_print
```

every time you install a new ruby, these two gems are installed into your global gemset. `default.gems` and `global.gems` files are usually overwritten during update of rvm.

第28章：更改默认时区

第28.1节：更改 Rails 时区并让 Active Record 将时间存储在该时区

```
# application.rb
config.time_zone = '美国东部时间（美国和加拿大）'
config.active_record.default_timezone = :local
```

第28.2节：更改Rails时区，但继续让Active Record以UTC保存到数据库中

```
# application.rb
config.time_zone = '美国东部时间（美国和加拿大）'
```


Chapter 28: Change default timezone

Section 28.1: Change Rails timezone AND have Active Record store times in this timezone

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

Section 28.2: Change Rails timezone, but continue to have Active Record save in the database in UTC

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

第29章：资源管线

资源管线提供了一个框架，用于连接和压缩JavaScript和CSS资源。它还
增加了使用其他语言和预处理器编写这些资源的能力，如CoffeeScript、Sass和ERB。它
允许应用中的资源自动与其他gem中的资源合并。例如，jquery-
rails包含jquery.js的副本，并启用Rails中的AJAX功能。

第29.1节：清单文件和指令

在assets初始化器（config/initializers/assets.rb）中，有几个文件被明确指定为预编译文件。

```
# 预编译额外的资源。
# application.coffee、application.scss以及app/assets文件夹中所有非JS/CSS文件已被添加。
# Rails.application.config.assets.precompile += %w( search.js )
```

在此示例中，`application.coffee` 和 `application.scss` 被称为“清单文件”。这些文件应
用于包含其他 JavaScript 或 CSS 资源。以下命令可用：

- require <path>：require 指令的功能类似于 Ruby 自带的 require。它提供了一种声明路径中某个文件依赖的方法，并确保该文件在源文件之前只加载一次。
- require_directory <path>：引入单个目录中的所有文件。它类似于 path/*，因为它不包含嵌套目录中的文件。

- require_tree <path>：引入目录中所有嵌套的文件。它的通配符等价为 path/**/*。
- require_self：使当前文件的主体在任何后续的 require 指令之前被插入。
  这在 CSS 文件中很有用，因为索引文件通常包含需要在其他依赖项加载之前定义的全局样式。

- stub <path>：从包含列表中移除某个文件
- depend_on <path>：允许声明对某个文件的依赖，但不包含该文件。此功能用于缓存目的。对依赖文件的任何更改都会使源文件的缓存失效。

一个 application.scss 文件可能如下所示：

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

另一个例子是application.coffee文件。这里包含了jquery和Turbolinks：

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

如果你不使用CoffeeScript，而是使用纯JavaScript，语法如下：

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```


Chapter 29: Asset Pipeline

The asset pipeline provides a framework to concatenate and minify or compress JavaScript and CSS assets. It also adds the ability to write these assets in other languages and pre-processors such as CoffeeScript, Sass and ERB. It allows assets in your application to be automatically combined with assets from other gems. For example, jquery-rails includes a copy of jquery.js and enables AJAX features in Rails.

Section 29.1: Manifest Files and Directives

In the `assets` initalizer (`config/initializers/assets.rb`) are a few files explicitly defined to be precompiled.

```
# Precompile additional assets.
# application.coffee, application.scss, and all non-JS/CSS in app/assets folder are already added.
# Rails.application.config.assets.precompile += %w( search.js )
```

In this example the `application.coffee` and `application.scss` are so called 'Manifest Files'. This files should be used to include other JavaScript or CSS assets. The following command are available:

- **`require <path>`**: The **`require`** directive functions similar to Ruby's own **`require`**. It provides a way to declare a dependency on a file in your path and ensures it's only loaded once before the source file.
- `require_directory <path>`: requires all the files inside a single directory. It's similar to `path/*` since it does not follow nested directories.
- `require_tree <path>`: requires all the nested files in a directory. Its glob equivalent is `path/**/*`.
- `require_self`: causes the body of the current file to be inserted before any subsequent **`require`** directives. Useful in CSS files, where it's common for the index file to contain global styles that need to be defined before other dependencies are loaded.
- `stub <path>`: remove a file from being included
- `depend_on <path>`: Allows you to state a dependency on a file without including it. This is used for caching purposes. Any changes made to the dependency file will invalidate the cache of the source file.

An `application.scss` file could look like:

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

Another example is the `application.coffee` file. Here with including `jquery` and `Turbolinks`:

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

If you don't use CoffeeScript, but plain JavaScript, the syntax would be:

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

第29.2节：Rake任务

默认情况下，sprockets-rails附带以下rake任务：

- `assets:clean[keep]`：删除旧的编译资源
- `assets:clobber`：删除编译资源
- `assets:environment`：加载资产编译环境
- `assets:precompile`：编译所有在`config.assets.precompile`中命名的资产

第29.3节：基本用法

资产管线的基本使用方式有两种：

1. 在开发模式下运行服务器时，它会自动即时预处理和准备你的资产。

2. 在生产模式下，你可能会用它来预处理、版本化、压缩和编译你的资产。
   你可以通过运行以下命令来实现：

   ```
   bundle exec rake assets:precompile
   ```


Section 29.2: Rake tasks

By default `sprockets-rails` is shipped with the following rake tasks:

- `assets:clean[keep]`: Remove old compiled assets
- `assets:clobber`: Remove compiled assets
- `assets:environment`: Load asset compile environment
- `assets:precompile`: Compile all the assets named in `config.assets.precompile`

Section 29.3: Basic Usage

There are two basic ways that the asset pipeline is used:

1. When running a server in development mode, it automatically pre-processes and prepares your assets on-the-fly.

2. In production mode, you'll probably use it to pre-process, versionize, and compress and compile your assets. You can do so by running the following command:

   ```
   bundle exec rake assets:precompile
   ```

第30章：升级Rails

第30.1节：从Rails 4.2升级到Rails 5.0

注意：在升级你的Rails应用之前，务必确保将代码保存到版本控制系统中，例如Git。

要从Rails 4.2升级到Rails 5.0，必须使用Ruby 2.2.2或更高版本。如果需要升级Ruby版本，完成后请打开你的Gemfile并修改以下行：

```
gem 'rails', '4.2.X'
```

到：

```
gem 'rails', '~> 5.0.0'
```

然后在命令行运行：

```
$ bundle update
```

现在使用以下命令运行更新任务：

```
$ rake rails:update
```

这将帮助你更新配置文件。系统会提示你是否覆盖文件，你有几个选项可输入：

- Y – 是，覆盖
- n – 否，不覆盖
- a – 全部，覆盖此文件及所有其他文件
- q – 退出，中止
- d – 差异，显示旧版本和新版本之间的区别
- h – 帮助

通常，您应该检查旧文件和新文件之间的差异，以确保没有出现任何不想要的更改。

Rails 5.0 ActiveRecord 模型继承自 ApplicationRecord，而不是 ActiveRecord::Base。ApplicationRecord 是所有模型的超类，类似于 ApplicationController 是控制器的超类。为了适应模型处理的新方式，您必须在 app/models/文件夹中创建一个名为 application_record.rb 的文件，然后编辑该文件内容为：

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0 对回调的处理也略有不同。返回 false 的回调不会中断回调链，这意味着后续的回调仍会执行，这与 Rails 4.2 不同。升级时，Rails 4.2 的行为将保持不变，但您可以通过添加以下内容切换到 Rails 5.0 的行为：

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

添加到 config/application.rb 文件中。您可以通过调用 throw(:abort) 来显式中断回调链。

Chapter 30: Upgrading Rails

Section 30.1: Upgrading from Rails 4.2 to Rails 5.0

Note: Before upgrading your Rails app, always make sure to save your code on a version control system, such as Git.

To upgrade from Rails 4.2 to Rails 5.0, you must be using Ruby 2.2.2 or newer. After upgrading your Ruby version if required, go to your Gemfile and change the line:

```
gem 'rails', '4.2.X'
```

to:

```
gem 'rails', '~> 5.0.0'
```

and on the command line run:

```
$ bundle update
```

Now run the update task using the command:

```
$ rake rails:update
```

This will help you to update configuration files. You will be prompted to overwrite files and you have several options to input:

- Y – yes, overwrite
- n – no, do not overwrite
- a – all, overwrite this and all others
- q – quit, abort
- d – diff, show the differences between the old and the new
- h – help

Typically, you should check the differences between the old and new files to make sure you aren't getting any unwanted changes.

Rails 5.0 `ActiveRecord` models inherit from `ApplicationRecord`, rather than `ActiveRecord::Base`. `ApplicationRecord` is the superclass for all models, similar to how `ApplicationController` is the superclass for controllers. To account for this new way in which models are handled, you must create a file in your `app/models/` folder called `application_record.rb` and then edit that file's contents to be:

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0 also handles callbacks slightly different. Callbacks that return **`false`** won't halt the callback chain, which means subsequent callbacks will still run, unlike Rails 4.2. When you upgrade, the Rails 4.2 behavior will remain, though you can switch to the Rails 5.0 behavior by adding:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

to the `config/application.rb` file. You can explicitly halt the callback chain by calling `throw(:abort)`.

在 Rails 5.0 中，ActiveJob 将继承自 ApplicationJob，而不是像 Rails 4.2 中继承自 ActiveJob::Base。要升级到 Rails 5.0，请在 app/jobs/ 文件夹中创建一个名为 application_job.rb 的文件。编辑该文件内容为：

```
class ApplicationJob < ActiveJob::Base
end
```

然后，您必须将所有作业改为继承自ApplicationJob，而不是ActiveJob::Base。

Rails 5.0的另一个重大变化不需要任何代码更改，但会改变您使用命令行操作Rails应用的方式。您将能够使用bin/rails，或者直接使用rails来运行任务和测试。例如，不再使用$ rake db:migrate，而是可以使用$ rails db:migrate。如果运行$ bin/rails，您可以查看所有可用命令。请注意，许多现在可以通过bin/rails运行的任务仍然可以使用rake来执行。

In Rails 5.0, `ActiveJob` will inherit from `ApplicationJob`, rather than `ActiveJob::Base` like in Rails 4.2. To upgrade to Rails 5.0, create a file called `application_job.rb` in the `app/jobs/` folder. Edit that file's contents to be:

```
class ApplicationJob < ActiveJob::Base
end
```

Then, you must change all of your jobs to inherit from `ApplicationJob` rather than `ActiveJob::Base`.

One of the other biggest changes of Rails 5.0 doesn't require any code changes, but will change the way you use the command line with your Rails apps. You will be able to use `bin/rails`, or just `rails`, to run tasks and tests. For example, instead of using `$ rake db:migrate`, you can now do `$ rails db:migrate`. If you run `$ bin/rails`, you can view all the available commands. Note that many of the tasks that can now be run with `bin/rails` still work using `rake`.

第31章：ActiveRecord锁定

第31.1节：乐观锁定

```
user_one = User.find(1)
user_two = User.find(1)

user_one.name = "John"
user_one.save
# 同时运行
user_two.name = "Doe"
user_two.save # 抛出ActiveRecord::StaleObjectError异常
```

第31.2节：悲观锁定

```
appointment = Appointment.find(5)
appointment.lock!
#没有其他用户可以读取此预约，
#他们必须等待锁被释放
预约。保存！
#锁定已释放，其他用户可以读取此账户
```


Chapter 31: ActiveRecord Locking

Section 31.1: Optimistic Locking

```
user_one = User.find(1)
user_two = User.find(1)

user_one.name = "John"
user_one.save
# Run at the same instance
user_two.name = "Doe"
user_two.save # Raises a ActiveRecord::StaleObjectError
```

Section 31.2: Pessimistic Locking

```
appointment = Appointment.find(5)
appointment.lock!
#no other users can read this appointment,
#they have to wait until the lock is released
appointment.save!
#lock is released, other users can read this account
```

第32章：调试

第32.1节：调试Rails应用程序

能够调试应用程序对于理解应用程序的逻辑和数据流程非常重要。它有助于解决逻辑错误，并提升编程体验和代码质量。两个流行的调试gem是debugger（适用于ruby 1.9.2和1.9.3）和byebug（适用于ruby >= 2.x）。

调试.rb文件，请按照以下步骤操作：

1. 将debugger或byebug添加到Gemfile的development组中
2. 运行bundle install
3. 将debugger或byebug作为断点添加
4.运行代码或发起请求
5. 查看 Rails 服务器日志在指定断点处停止
6. 此时你可以像使用rails console一样使用服务器终端，检查变量和params
7. 要执行下一条指令，输入next并按enter
8. 要跳出当前步骤，输入c并按enter

如果你想调试.html.erb文件，断点将以<% debugger %>形式添加

第32.2节：快速调试 Ruby on Rails + 初学者建议

通过抛出异常进行调试比通过print日志语句仔细查看要简单得多，对于大多数错误，通常比打开 irb 调试器如pry或byebug要快得多。这些工具不应作为你的第一步。

快速调试 Ruby/Rails：

1. 快速方法：引发一个Exception然后调用.inspect查看其结果调试Ru

by（尤其是Rails）代码的最快方法是在代码执行路径中raise一个异常，同时调用方法或对象（例如foo）的.inspect：

```
raise foo.inspect
```

在上述代码中，raise会触发一个Exception，停止代码执行，并返回一条错误信息，方便地包含了你正在调试的行上对象/方法（即foo）的.inspect信息。

该技巧适用于快速检查一个对象或方法（例如它是否为nil？），以及立即确认在给定上下文中某行代码是否被执行。

2. 备用方案：使用Ruby的IRB调试器，如byebug或pry

只有在你获得了代码执行流程状态的信息后，才应考虑使用Ruby的gem IRB调试器，如pry或byebug，深入探查执行路径中对象的状态。
执行路径。

在Rails中使用byebug gem进行调试：

Chapter 32: Debugging

Section 32.1: Debugging Rails Application

To be able to debug an application is very important to understand the flow of an application's logic and data. It helps solving logical bugs and adds value to the programming experience and code quality. Two popular gems for debugging are debugger (for ruby 1.9.2 and 1.9.3) and byebug (for ruby >= 2.x).

For debugging `.rb` files, follow these steps:

1. Add `debugger` or `byebug` to the `development` group of `Gemfile`
2. Run `bundle install`
3. Add `debugger` or `byebug` as the breakpoint
4. Run the code or make request
5. See the rails server log stopped at the specified breakpoint
6. At this point you can use your server terminal just like `rails console` and check the values of variable and params
7. For moving to next instruction, type `next` and press `enter`
8. For stepping out type `c` and press `enter`

If you want to debug `.html.erb` files, break point will be added as `<% debugger %>`

Section 32.2: Debugging Ruby on Rails Quickly + Beginner advice

Debugging by raising exceptions is *far easier* than squinting through `print` log statements, and for most bugs, its generally *much faster* than opening up an irb debugger like `pry` or `byebug`. Those tools should not be your first step.

Debugging Ruby/Rails Quickly:

1. Fast Method: Raise an `Exception` then and `.inspect` its result

The *fastest* way to debug Ruby (especially Rails) code is to `raise` an exception along the execution path of your code while calling `.inspect` on the method or object (e.g. `foo`):

```
raise foo.inspect
```

In the above code, ***`raise`*** triggers an ***`Exception`*** that *halts execution of your code*, and returns an error message that conveniently contains `.inspect` information about the object/method (i.e. `foo`) on the line that you're trying to debug.

This technique is useful for *quickly* examining an object or method (*e.g. is it **`nil`**?*) and for immediately confirming whether a line of code is even getting executed at all within a given context.

2. Fallback: Use a ruby *IRB* debugger like `byebug` or `pry`

Only after you have information about the state of your codes execution flow should you consider moving to a ruby gem irb debugger like `pry` or `byebug` where you can delve more deeply into the state of objects within your execution path.

To use the `byebug` gem for debugging in Rails:

1. 在你的Gemfile的development组中添加gem 'byebug'
2. 运行bundle install
3. 然后使用时，在你想要检查的代码执行路径中插入短语byebug。

当执行这个byebug变量时，会打开一个 Ruby IRB 会话，允许你直接访问代码执行到该点时对象的状态。

像 Byebug 这样的 IRB 调试器对于深入分析代码执行状态非常有用。然而，与抛出错误相比，它们是更耗时的过程，因此在大多数情况下不应作为你的第一步。

一般初学者建议

当你试图调试问题时，一个好的建议是始终：认真阅读错误信息（RTFM）这意味着在采取行动之前，要仔细且完整

地阅读错误信息，以便你理解它想告诉你的内容。调试时，阅读错误信息时请按以下顺序问自己这些心理问题：

1. *错误引用的是哪个类？（例如，我的对象类是否正确，或者我的对象是否为nil？）*
2. *错误引用的是哪个方法？（例如，方法中是否有拼写错误；我能否在该类型/类的对象上调用此方法？）*
3. 最后，结合我从前两个问题推断出的信息，我应该调查哪些代码行？（记住：堆栈跟踪中的最后一行代码不一定是问题所在。）

在堆栈跟踪中，特别注意来自您项目的代码行（例如以...开头的行）app/...（如果你使用的是 Rails）。99% 的情况下问题出在你自己的代码上。

为了说明为什么按这个顺序解释很重要...

例如，一个让许多初学者困惑的 Ruby 错误信息：

你执行的代码在某个时刻执行如下：

```
@foo = Foo.new

...

@foo.bar
```

然后你得到一个错误提示：

```
undefined method "bar" for Nil:nilClass
```

初学者看到这个错误，认为问题是方法bar是未定义的。其实不是。在这个错误中真正重要的部分是：

```
for Nil:nilClass
```

对于Nil:nilClass意味着@foo是Nil！@foo不是Foo的实例变量！你有一个对象是Nil。当你看到这个错误时，这只是ruby在告诉你方法bar不存在于Nil类的对象上。
（显然！因为我们试图对Foo类的对象使用方法，而不是Nil类的对象）。

不幸的是，由于这个错误的写法（undefined method "bar" for Nil:nilClass），很容易让人误以为这个错误是因为bar未定义。当不仔细阅读时，这个错误会导致初学者错误地去深入Foo类中bar方法的细节，完全忽略了错误中提示对象属于错误类的部分（在本例中是nil）。

1. Add `gem 'byebug'` inside the *development* group in your *Gemfile*
2. Run `bundle install`
3. Then to use, insert the phrase `byebug` inside the execution path of the code you want examined.

This `byebug` variable when executed will open up an ruby IRB session of your code, giving you direct access to the state of objects as they are at that point in the code's execution.

IRB debuggers like Byebug are useful for deeply analyzing the state of your code as it executes. However, they are more time consuming procedure compared to raising errors, so in most situations they should not be your first step.

General Beginner Advice

When you are trying to debug a problem, good advice is to always: **Read The !@#$ing Error Message (RTFM)**

That means reading error messages *carefully* and *completely* before acting so that you *understand what it's trying to tell you.* When you debug, ask the following mental questions, *in this order*, when reading an error message:

1. What ***class*** does the error reference? (i.e. *do I have the correct object class or is my object* ***nil****?*)
2. What ***method*** does the error reference? (i.e. *is their a type in the method; can I call this method on this type/class of object?*)
3. Finally, using what I can infer from my last two questions, what ***lines of code*** should I investigate? (remember: the last line of code in the stack trace is not necessarily where the problem lies.)

In the stack trace pay particular attention to lines of code that come from your project (e.g. lines starting with `app/...` if you are using Rails). 99% of the time the problem is with your own code.

To illustrate why interpreting *in this order* is important...

E.g. a Ruby error message that confuses many beginners:

You execute code that at some point executes as such:

```
@foo = Foo.new

...

@foo.bar
```

and you get an error that states:

```
undefined method "bar" for Nil:nilClass
```

Beginners see this error and think the problem is that the method `bar` is *undefined*. **It's not.** In this error the real part that matters is:

```
for Nil:nilClass
```

`for Nil:nilClass` means that `@foo` is Nil! `@foo` is not a `Foo` instance variable! You have an object that is `Nil`. When you see this error, it's simply ruby trying to tell you that the method `bar` doesn't exist for objects of the class `Nil`. (well duh! since we are trying to use a method for an object of the class `Foo` not `Nil`).

Unfortunately, due to how this error is written (`undefined method "bar" for Nil:nilClass`) its easy to get tricked into thinking this error has to do with `bar` being `undefined`. When not read carefully this error causes beginners to mistakenly go digging into the details of the `bar` method on `Foo`, entirely missing the part of the error that hints that

这是一个很容易避免的错误，只需完整阅读错误信息即可。

总结：

在开始调试之前，务必仔细阅读完整的错误信息。这意味着：首先检查错误信息中对象的类类型，然后是它的方法，再开始调查任何堆栈跟踪或你认为错误可能发生的代码行。这5秒钟的时间可以为你节省5小时的挫败感。

简而言之：不要眯着眼看打印日志：应抛出异常。通过仔细阅读错误信息来避免陷入无谓的调试。

第32.3节：使用pry调试ruby-on-rails应用程序

pry是一个强大的工具，可以用来调试任何ruby应用程序。使用这个gem设置ruby-on-rails应用程序非常简单直接。

设置

开始使用 pry 调试您的应用程序

- 在应用程序的Gemfile中添加gem'pry'并运行bundle安装

```
group :development, :test do
  gem 'pry'
end
```

- 在终端控制台中导航到应用程序的根目录并运行bundle install。你就可以在应用程序的任何地方开始使用它了。

使用

在应用程序中使用pry就是在你想调试时检查的断点处包含binding.pry。
你可以在应用程序中任何由ruby解释器解释的地方添加binding.pry断点（任何app/controllers、app/models、app/views文件）

i) 调试控制器

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def show
use_id = params[:id]
    //断点检查如果动作接收到的参数符合预期
    binding.pry
    @user = User.find(user_id)
    respond_to do |format|
      format.html
    结束
  结束
end
```

在此示例中，当你尝试访问路由到
UsersController的show动作的页面时，rails服务器会在断点处暂停并进入pry控制台。你可以检查params对象并在该断点处对User模型进行ActiveRecord查询

the object is of the wrong class (in this case: nil). It's a mistake that's easily avoided by reading error messages in their entirety.

Summary:

Always carefully **read the *entire* error message** before beginning any debugging. That means: Always check the ***class*** type of an object in an error message *first*, then its ***methods***, *before* you begin sleuthing into any stacktrace or line of code where you think the error may be occurring. Those 5 seconds can save you 5 hours of frustration.

tl;dr: Don't squint at print logs: raise exceptions instead. Avoid rabbit holes by reading errors carefully before debugging.

Section 32.3: Debugging ruby-on-rails application with pry

pry is a powerful tool that can be used to debug any ruby application. Setting up a ruby-on-rails application with this gem is very easy and straightforward.

Setup

To start debugging your application with pry

- Add `gem 'pry'` to the application's `Gemfile` and bundle it

```
group :development, :test do
  gem 'pry'
end
```

- Navigate to the application's root directory on terminal console and run `bundle install`. You're all set to start using it anywhere on your application.

Use

Using pry in your application is just including **`binding.pry`** on the breakpoints you want to inspect while debugging. You can add **`binding.pry`** breakpoints anywhere in your application that is interpreted by ruby interpreter (any app/controllers, app/models, app/views files)

i) Debugging a Controller

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def show
    use_id = params[:id]
    // breakpoint to inspect if the action is receiving param as expected
    binding.pry
    @user = User.find(user_id)
    respond_to do |format|
      format.html
    end
  end
end
```

In this example, the rails server pauses with a pry console at the break-point when you try to visit a page routing to `show` action on `UsersController`. You can inspect `params` object and make ActiveRecord query on `User` model from that breakpoint

ii) 调试视图

app/views/users/show.html.haml

```
%table
  %tbody
    %tr
      %td ID
      %td= @user.id
    %tr
      %td email
      %td= @user.email
    %tr
      %td logged in ?
      %td
        - binding.pry
        - if @user.logged_in?
          %p= "Logged in"
        - else
          %p= "Logged out"
```

在此示例中，当 Rails 服务器在将 users/show 页面预编译后发送回客户端浏览器时，断点会在 pry 控制台暂停。此断点允许调试正确性
@user.logged_in? 当它出现异常行为时。

ii) 调试模型

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

在此示例中，可以使用断点调试User模型的实例方法full_name，当该方法
从应用程序的任何位置被调用时。

总之，pry 是一个功能强大的 Rails 应用调试工具，设置简单，调试指南直观。试试看吧。

第32.4节：在你的IDE中调试

每个优秀的IDE都提供了一个GUI，用于交互式调试 Ruby（因此也包括 Rails）应用程序，你可以添加断点、监视、异常时自动暂停，并允许你逐步、逐行地跟踪代码执行。

例如，看看最好的 Ruby IDE 之一 RubyMine 的调试功能，如图所示

ii) Debugging a View

app/views/users/show.html.haml

```
%table
  %tbody
    %tr
      %td ID
      %td= @user.id
    %tr
      %td email
      %td= @user.email
    %tr
      %td logged in ?
      %td
        - binding.pry
        - if @user.logged_in?
          %p= "Logged in"
        - else
          %p= "Logged out"
```

In this example, the break-point pauses with pry console when the `users/show` page is pre-compiled in the rails server before sending it back to the client's browser. This break-point allows to debug correctness of `@user.logged_in?` when it is misbehaving.

ii) Debugging a Model

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

In this example, the break-point can be used to debug `User` model's instance method `full_name` when this method is called from anywhere in the application.

In conclusion, pry is a powerful debugging tool for rails application with easy setup and straightforward debugging guideline. Give this a try.

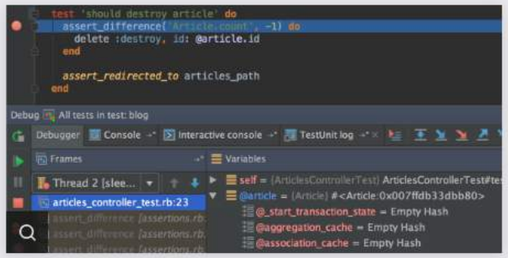
Section 32.4: Debugging in your IDE

Every good IDE provides a GUI for interactively debugging Ruby (and thus Rails) applications where you can add breakpoints, watches, auto pausing on exception and allows you to follow the code execution even step by step, line by line.

For example, take a look at one of the best Ruby IDE's, RubyMine's debugging features on the picture

Powerful Debugger

RubyMine brings a clever debugger with a graphical UI for Ruby, JS, and CoffeeScript. Set breakpoints and run your code step by step with all the information at your fingertips.



Convenient user interface

- Look under the hood of any code and see what's going on—thanks to Frames, Variables and Watches views.
- The UI is fully customizable letting you arrange, resize and float the views, select toolbar commands, etc. You can also choose whether to ignore non-project code while stepping or not.
- The debugger UI is also tightly integrated with the code editor: you can navigate between the debugger and the code, view breakpoint information, etc.
- You also get the complete coding assistance and highlighting in all debugger views.

Smart, flexible breakpoints

- Place a breakpoint on a line of code and define the hit conditions—a set of Boolean expressions that are evaluated to determine whether to stop the code execution or not.
- If you have multiple breakpoints in your code, you can set dependencies between them to define the order in which they can be hit.
- Setting a breakpoint is just a matter of a single mouse click on the gutter or invoking a shortcut.
- Breakpoints are also available in Rails views, so you can use them for debugging Rails code as well.

Debugging JavaScript and Node.js

- RubyMine provides an advanced built-in debugger for your JavaScript code, which works with Google Chrome.
- You can easily debug ECMAScript 6, TypeScript or CoffeeScript code, relying on RubyMine debugger's support for source maps.
- A full-featured debugger for Node.js works right out of the box as well. Use it to debug apps running locally or on a remote machine.

Built-in expression evaluator

Evaluate any expression while your debugging session is paused. Type an expression or a code fragment, with coding assistance available in the dialog. All expressions are evaluated against the current context.

Dedicated Watches view

Track any number of expressions, variables, fields and object instances in the current stack frame context. The data are updated dynamically as you progress through your debugging session.

Frames and call stack

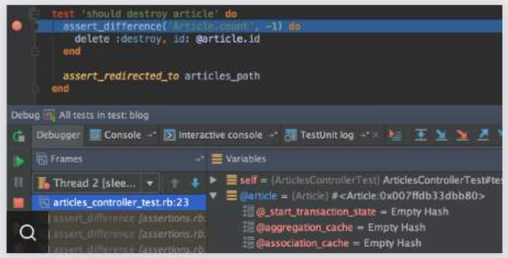
When a breakpoint is hit or code execution is suspended, you can use the Frames panel to examine the current threads, their state, call stack, methods, and variables along with their values.

Remote debugging

As you connect to a remote host and run a debugging session, RubyMine keeps mapping between the local sources and the sources on the remote side. Several debug processes can be launched simultaneously.

Powerful Debugger

RubyMine brings a clever debugger with a graphical UI for Ruby, JS, and CoffeeScript. Set breakpoints and run your code step by step with all the information at your fingertips.



Convenient user interface

- Look under the hood of any code and see what's going on—thanks to Frames, Variables and Watches views.
- The UI is fully customizable letting you arrange, resize and float the views, select toolbar commands, etc. You can also choose whether to ignore non-project code while stepping or not.
- The debugger UI is also tightly integrated with the code editor: you can navigate between the debugger and the code, view breakpoint information, etc.
- You also get the complete coding assistance and highlighting in all debugger views.

Smart, flexible breakpoints

- Place a breakpoint on a line of code and define the hit conditions—a set of Boolean expressions that are evaluated to determine whether to stop the code execution or not.
- If you have multiple breakpoints in your code, you can set dependencies between them to define the order in which they can be hit.
- Setting a breakpoint is just a matter of a single mouse click on the gutter or invoking a shortcut.
- Breakpoints are also available in Rails views, so you can use them for debugging Rails code as well.

Debugging JavaScript and Node.js

- RubyMine provides an advanced built-in debugger for your JavaScript code, which works with Google Chrome.
- You can easily debug ECMAScript 6, TypeScript or CoffeeScript code, relying on RubyMine debugger's support for source maps.
- A full-featured debugger for Node.js works right out of the box as well. Use it to debug apps running locally or on a remote machine.

Built-in expression evaluator

Evaluate any expression while your debugging session is paused. Type an expression or a code fragment, with coding assistance available in the dialog. All expressions are evaluated against the current context.

Dedicated Watches view

Track any number of expressions, variables, fields and object instances in the current stack frame context. The data are updated dynamically as you progress through your debugging session.

Frames and call stack

When a breakpoint is hit or code execution is suspended, you can use the Frames panel to examine the current threads, their state, call stack, methods, and variables along with their values.

Remote debugging

As you connect to a remote host and run a debugging session, RubyMine keeps mapping between the local sources and the sources on the remote side. Several debug processes can be launched simultaneously.

第33章：使用Rails配置Angular

第33.1节：使用Rails的Angular入门

步骤1：创建一个新的Rails应用

```
gem install rails -v 4.1
rails new angular_example
```

步骤2：移除Turbolinks

移除turbolinks需要从Gemfile中删除它。

```
gem 'turbolinks'
```

从app/assets/javascripts/application.js中移除require：

```
//= require turbolinks
```

步骤3：将AngularJS添加到资产管线中

为了让Angular能与Rails的资产管线一起工作，我们需要在Gemfile中添加：

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

现在运行命令

```
bundle install
```

添加 bower，以便我们可以安装 AngularJS 依赖：

```
rails g bower_rails:initialize json
```

向 bower.json 添加 Angular：

```
{
  "name": "bower-rails 生成的依赖",

  "dependencies": {

    "angular": "latest",
    "angular-resource": "latest",
    "bourbon": "latest",
    "angular-bootstrap": "latest",
    "angular-ui-router": "latest"
  }
}
```

既然 bower.json 已经配置了正确的依赖，现在让我们安装它们：

```
bundle exec rake bower:install
```

步骤4：组织Angular应用

在app/assets/javascript/angular-app/中创建以下文件夹结构：

```
templates/
```


Chapter 33: Configure Angular with Rails

Section 33.1: Angular with Rails 101

Step 1: Create a new Rails app

```
gem install rails -v 4.1
rails new angular_example
```

Step 2: Remove Turbolinks

Removing turbolinks requires removing it from the Gemfile.

```
gem 'turbolinks'
```

Remove the `require` from `app/assets/javascripts/application.js`:

```
//= require turbolinks
```

Step 3: Add AngularJS to the asset pipeline

In order to get Angular to work with the Rails asset pipeline we need to add to the Gemfile:

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

Now run the command

```
bundle install
```

Add `bower` so that we can install the AngularJS dependency:

```
rails g bower_rails:initialize json
```

Add Angular to `bower.json`:

```
{
  "name": "bower-rails generated dependencies",

  "dependencies": {

    "angular": "latest",
    "angular-resource": "latest",
    "bourbon": "latest",
    "angular-bootstrap": "latest",
    "angular-ui-router": "latest"
  }
}
```

Now that `bower.json` is setup with the right dependencies, let's install them:

```
bundle exec rake bower:install
```

Step 4: Organize the Angular app

Create the following folder structure in `app/assets/javascript/angular-app/`:

```
templates/
```

```
modules/
filters/
directives/
models/
services/
controllers/
```

在app/assets/javascripts/application.js中，添加**require**来引入Angular、模板助手和Angular应用的文件结构。如下所示：

```
//= require jquery
//= require jquery_ujs

//= require angular
//= require angular-rails-templates
//= require angular-app/app

//= require_tree ./angular-app/templates
//= require_tree ./angular-app/modules
//= require_tree ./angular-app/filters
//= require_tree ./angular-app/directives
//= require_tree ./angular-app/models
//= require_tree ./angular-app/services
//= require_tree ./angular-app/controllers
```

步骤5：启动Angular应用

创建app/assets/javascripts/angular-app/app.js.coffee文件：

```
@app = angular.module('app', [ 'templates' ])

@app.config([ '$httpProvider', ($httpProvider)->
$httpProvider.defaults.headers.common['X-CSRF-Token'] = $('meta[name=csrftoken]').attr('content')
])  @app.run(->   console.log 'angular app running' )
```

在app/assets/javascripts/angular-app/modules/example.js.coffee.erb中创建一个Angular模块：

```
@exampleApp = angular.module('app.exampleApp', [     # 这里添加额外依赖    ])   .run(->
  console.log 'exampleApp running'    )
```

在app/assets/javascripts/angular-app/controllers/exampleCtrl.js.coffee中为该应用创建一个Angular控制器：

```
angular.module('app.exampleApp').controller("ExampleCtrl", [   '$scope',   ($scope)->
console.log 'ExampleCtrl running'      $scope.exampleValue = "Hello angular and rails"  ])
```

现在在Rails中添加一个路由，将控制权交给Angular。在config/routes.rb中：

```
Rails.application.routes.draw do    get 'example' => 'example#index' end
```

生成响应该路由的Rails控制器：

```
rails g controller Example
```

在app/controllers/example_controller.rb中：

```
class ExampleController < ApplicationController
    def index
```



```
modules/
filters/
directives/
models/
services/
controllers/
```

In app/assets/javascripts/application.js, add **require** for Angular, the template helper, and the Angular app file structure. Like this:

```
//= require jquery
//= require jquery_ujs

//= require angular
//= require angular-rails-templates
//= require angular-app/app

//= require_tree ./angular-app/templates
//= require_tree ./angular-app/modules
//= require_tree ./angular-app/filters
//= require_tree ./angular-app/directives
//= require_tree ./angular-app/models
//= require_tree ./angular-app/services
//= require_tree ./angular-app/controllers
```

Step 5: Bootstrap the Angular app

Create app/assets/javascripts/angular-app/app.js.coffee:

```
@app = angular.module('app', [ 'templates' ])

@app.config([ '$httpProvider', ($httpProvider)->
$httpProvider.defaults.headers.common['X-CSRF-Token'] = $('meta[name=csrftoken]').attr('content')
])  @app.run(->   console.log 'angular app running' )
```

Create an Angular module at app/assets/javascripts/angular-app/modules/example.js.coffee.erb:

```
@exampleApp = angular.module('app.exampleApp', [     # additional dependencies here    ])   .run(->
  console.log 'exampleApp running'    )
```

Create an Angular controller for this app at app/assets/javascripts/angular-app/controllers/exampleCtrl.js.coffee:

```
angular.module('app.exampleApp').controller("ExampleCtrl", [   '$scope',   ($scope)->
console.log 'ExampleCtrl running'      $scope.exampleValue = "Hello angular and rails"  ])
```

Now add a route to Rails to pass control over to Angular. Inconfig/routes.rb:

```
Rails.application.routes.draw do    get 'example' => 'example#index' end
```

Generate the Rails controller to respond to that route:

```
rails g controller Example
```

In app/controllers/example_controller.rb:

```
class ExampleController < ApplicationController
    def index
```

```
    结束
结束
```

在视图中，我们需要指定哪个Angular应用和哪个Angular控制器将驱动此页面。因此在app/views/example/index.html.erb中：

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>

  <p>来自ExampleCtrl的值：</p>
  <p>{{ exampleValue }}</p>

</div>
```

要查看该应用，请启动你的Rails服务器并访问 http://localhost:3000/example。


```
    end
end
```

In the view, we need to specify which Angular app and which Angular controller will drive this page. So in app/views/example/index.html.erb:

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>

  <p>Value from ExampleCtrl:</p>
  <p>{{ exampleValue }}</p>

</div>
```

To view the app, start your Rails server and visit http://localhost:3000/example.

第34章：Rails日志记录器

第34.1节：Rails.logger

始终使用Rails.logger.{debug|info|warn|error|fatal}而不是puts。这样可以让你的日志符合标准日志格式，带有时间戳和级别，方便你选择是否在特定环境中显示它们。你可以在log/目录下看到你的Rails应用环境名称对应的独立日志文件，比如：development.log或production.log或staging.log

你可以使用LogRotate轻松地轮转Rails生产日志。你只需进行如下简单配置用你喜欢的Linux编辑器vim或nano打

开/etc/logrotate.conf文件，并在文件底部添加以下代码。

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
missingok
   rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

那么，它是如何工作的这非常简单。配置的每一部分功能如下：

- daily – 每天轮转日志文件。你也可以改用每周或每月。
- missingok – 如果日志文件不存在，则忽略它
- rotate 7 – 只保留7天的日志
- compress – 轮转时对日志文件进行GZip压缩
- delaycompress – 先轮转文件，第二天再压缩，这样可以确保不会干扰Rails服务器

- notifempty – 如果日志为空，则不旋转文件copytruncat
- e – 复制日志文件然后清空它。这确保了 Rails 写入的日志文件始终存在，因此不会因为文件实际上没有变化而出现问题。如果不使用此选项，每次都需要重启 Rails 应用程序。

运行 Logrotate 由于我们刚刚编写了这个配置，你需要测试它。

要手动运行 logrotate，只需执行：`sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

就这么简单。

Chapter 34: Rails logger

Section 34.1: Rails.logger

Always use `Rails.logger.{debug|info|warn|error|fatal}` rather than `puts`. This allows your logs to fit into the standard log format, have a timestamp and have a level so you choose whether they are important enough to be shown in a specific environment. You can see the separate log files for your application under `log/` directory with your rails app environment name. like: `development.log` or `production.log` or `staging.log`

You can easily rotating rails production logs with LogRotate.You just have to do small configuration as below

Open `/etc/logrotate.conf` with your favourite linux editor `vim` or `nano` and add the below code in this file at bottom.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

So, **How It Works** This is fantastically easy. Each bit of the configuration does the following:

- **daily** – Rotate the log files each day. You can also use weekly or monthly here instead.
- **missingok** – If the log file doesn't exist,ignore it
- **rotate 7** – Only keep 7 days of logs around
- **compress** – GZip the log file on rotation
- **delaycompress** – Rotate the file one day, then compress it the next day so we can be sure that it won't interfere with the Rails server
- **notifempty** – Don't rotate the file if the logs are empty
- **copytruncate** – Copy the log file and then empties it. This makes sure that the log file Rails is writing to always exists so you won't get problems because the file does not actually change. If you don't use this, you would need to restart your Rails application each time.

Running Logrotate Since we just wrote this configuration, you want to test it.

To run logrotate manually, just do: `sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

That's it.

第35章：Prawn PDF

第35.1节：高级示例

这是带示例的高级方法

```
class FundsController < ApplicationController

  def index
    @funds = Fund.all_funds(current_user)
  end

  def show
    @fund = Fund.find(params[:id])
    respond_to do |format|
      format.html
      format.pdf do
pdf = FundsPdf.new(@fund, view_context)
        send_data pdf.render, filename:
        "fund_#{@fund.created_at.strftime("%d/%m/%Y")}.pdf",
        type: "application/pdf"
      结束
    结束
  end
结束
```

在上述代码中，我们有这一行FundsPdf.new(@fund, view_context)。这里我们用@fund实例和view_context初始化了FundsPdf类，以便在FundsPdf中使用辅助方法。FundsPdf类大致如下

```
class FundPdf < Prawn::Document

  def initialize(fund, view)
    super()
    @fund = fund
    @view = view
    upper_half
    lower_half
  end

  def upper_half
logopath =  "#{Rails.root}/app/assets/images/logo.png"
    image logopath, :width => 197, :height => 91
向下移动 10
绘制文本 "收据", :at => [220, 575], 大小: 22
    向下移动 80
text "Hello #{@invoice.customer.profile.first_name.capitalize},"
  end

  def thanks_message
move_down 15
text "感谢您的订单。请打印此收据作为
    您订单的确认。",
    :indent_paragraphs => 40, :size => 13
  end
end
```

这是使用 Prawn gem 通过类生成 PDF 的最佳方法之一。

Chapter 35: Prawn PDF

Section 35.1: Advanced Example

This is the advanced approach with example

```
class FundsController < ApplicationController

  def index
    @funds = Fund.all_funds(current_user)
  end

  def show
    @fund = Fund.find(params[:id])
    respond_to do |format|
      format.html
      format.pdf do
        pdf = FundsPdf.new(@fund, view_context)
        send_data pdf.render, filename:
        "fund_#{@fund.created_at.strftime("%d/%m/%Y")}.pdf",
        type: "application/pdf"
      end
    end
  end
end
```

I above code we have this line `FundsPdf.new(@fund, view_context)`. Here we are initializing FundsPdf class with @fund instance and view_context to use helper methods in FundsPdf. FundsPdf wuld look like this

```
class FundPdf < Prawn::Document

  def initialize(fund, view)
    super()
    @fund = fund
    @view = view
    upper_half
    lower_half
  end

  def upper_half
    logopath =  "#{Rails.root}/app/assets/images/logo.png"
    image logopath, :width => 197, :height => 91
    move_down 10
    draw_text "Receipt", :at => [220, 575], size: 22
    move_down 80
    text "Hello #{@invoice.customer.profile.first_name.capitalize},"
  end

  def thanks_message
    move_down 15
    text "Thank you for your order.Print this receipt as
    confirmation of your order.",
    :indent_paragraphs => 40, :size => 13
  end
end
```

This is one of the best approach to generate PDF with classes using Prawn gem.

第35.2节：基本示例

您需要在 mime_types.rb 中添加 Gem 和 PDF MIME 类型，因为我们需要通知 Rails 关于 PDF 的 MIME 类型。

之后我们可以通过以下基本方式使用 Prawn 生成 PDF

这是基本的赋值示例

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

我们可以使用隐式块来实现

```
Prawn::Document.generate("implicit.pdf") do
 text "Hello World"
end
```

使用显式块

```
Prawn::Document.generate("explicit.pdf") do |pdf|
 pdf.text "Hello World"
end
```


Section 35.2: Basic Example

You need to add Gem and PDF MIME:Type inside mime_types.rb as we need to notify rails about PDF mime type.

After that we can generate Pdf with Prawn in following basic ways

This is the basic assignment

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

We can do it with Implicit Block

```
Prawn::Document.generate("implicit.pdf") do
 text "Hello World"
end
```

With Explicit Block

```
Prawn::Document.generate("explicit.pdf") do |pdf|
 pdf.text "Hello World"
end
```

第36章：Rails API

第36.1节：创建仅API应用程序

要构建一个作为API服务器的Rails应用程序，可以从Rails 5中更有限的Rails子集开始。

生成一个新的Rails API应用程序：

```
rails new my_api --api
```

--api的作用是移除构建API时不需要的功能。这包括会话、Cookie、
资源以及任何使Rails在浏览器上运行的内容。

它还将配置生成器，使其在生成新资源时不生成视图、助手和资源文件。

当你比较网页应用中的ApplicationController和API应用中的ApplicationController时，你会发现网页版本继承自ActionController::Base，而API版本继承自ActionController::API，后者包含的功能子集要小得多。

Chapter 36: Rails API

Section 36.1: Creating an API-only application

To build a Rails application that will be an API server, you can start with a more limited subset of Rails in Rails 5.

To generate a new Rails API app:

```
rails new my_api --api
```

What `--api` does is to remove functionality that is not needed when building an API. This includes sessions, cookies, assets, and anything that makes Rails work on a browser.

It will also configure the generators so that they don't generate views, helpers, and assets when generating a new resource.

When you compare the `ApplicationController` on a web app versus an API app, you will see that the web version extends from `ActionController::Base`, whereas the API version extends from `ActionController::API`, which includes a much smaller subset of functionality.

第37章：在Heroku上部署Rails应用

第37.1节：部署你的应用

确保你处于包含Rails应用的目录中，然后在Heroku上创建一个应用。

```
$ heroku create example
正在创建 example... 完成
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

输出中的第一个URL，http://example.herokuapp.com，是应用可访问的位置。第二个URL，git@heroku.com:example.git，是远程git仓库的URL。

此命令应仅在已初始化的git仓库中使用。heroku create命令会自动添加一个名为“heroku”的git远程，指向该URL。

应用名称参数（“example”）是可选的。如果未指定应用名称，将会生成一个随机名称。
由于 Heroku 应用名称处于全局命名空间中，您可以预期常见名称，如“blog”或“wiki”，
很可能已经被占用。通常从默认名称开始，之后再重命名应用会更简单。

接下来，部署您的代码：

```
$ git push heroku master
remote: 正在压缩源文件... 完成。
remote: 正在构建源代码：
remote:
remote: -----> 检测到 Ruby 应用
remote: -----> 正在编译 Ruby/Rails
remote: -----> 使用的 Ruby 版本：ruby-2.3.1
remote: -----> 使用 bundler 1.11.2 安装依赖
remote:        正在运行：bundle install --without development:test --path vendor/bundle --binstubs
vendor/bundle/bin -j4 --deployment
remote:        警告：当前运行的 Bundler 版本低于创建锁文件时的版本。 建议您通过运行 `gem install bundler`升级到最新
版本的 Bundler。
remote:        正在从 https://rubygems.org/ 获取 gem 元数据 ..........
remote:        正在从 https://rubygems.org/ 获取版本元数据...
remote:        正在从 https://rubygems.org/ 获取依赖元数据..
remote:        正在安装 concurrent-ruby 1.0.2
remote:        正在安装 i18n 0.7.0
remote:        正在安装 rake 11.2.2
remote:        正在安装 minitest 5.9.0
remote:        正在安装 thread_safe 0.3.5
remote:        正在安装 builder 3.2.2
remote:        正在安装 mini_portile2 2.1.0
remote:        正在安装 erubis 2.7.0
remote:        正在安装 pkg-config 1.1.7
remote:        正在安装 rack 2.0.1
remote:        正在安装带有本地扩展的 nio4r 1.2.1
remote:        正在安装 websocket-extensions 0.1.2
remote:        正在安装 mime-types-data 3.2016.0521
remote:        正在安装 arel 7.0.0
remote:        正在安装 coffee-script-source 1.10.0
remote:        正在安装 execjs 2.7.0
remote:        正在安装 method_source 0.8.2
remote:        正在安装 thor 0.19.1
remote:        正在安装 multi_json 1.12.1
```


Chapter 37: Deploying a Rails app on Heroku

Section 37.1: Deploying your application

Make sure you are in the directory that contains your Rails app, then create an app on Heroku.

```
$ heroku create example
Creating ☐ example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

The first URL of the ouput, http://example.herokuapp.com, is the location the app is available at. The second URL, git@heroku.com:example.git, is the remote git repository URL.

This command should only be used on an initialized git repository. The heroku create command automatically adds a git remote named “heroku” pointing at this URL.

The app name argument (“example”) is optional. If no app name is specified, a random name will be generated. Since Heroku app names are in a global namespace, you can expect that common names, like “blog” or “wiki”, will already be taken. It’s often easier to start with a default name and rename the app later.

Next, deploy your code:

```
$ git push heroku master
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.3.1
remote: -----> Installing dependencies using bundler 1.11.2
remote:        Running: bundle install --without development:test --path vendor/bundle --binstubs
vendor/bundle/bin -j4 --deployment
remote:        Warning: the running version of Bundler is older than the version that created the
lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install bundler`.
remote:        Fetching gem metadata from https://rubygems.org/..........
remote:        Fetching version metadata from https://rubygems.org/...
remote:        Fetching dependency metadata from https://rubygems.org/..
remote:        Installing concurrent-ruby 1.0.2
remote:        Installing i18n 0.7.0
remote:        Installing rake 11.2.2
remote:        Installing minitest 5.9.0
remote:        Installing thread_safe 0.3.5
remote:        Installing builder 3.2.2
remote:        Installing mini_portile2 2.1.0
remote:        Installing erubis 2.7.0
remote:        Installing pkg-config 1.1.7
remote:        Installing rack 2.0.1
remote:        Installing nio4r 1.2.1 with native extensions
remote:        Installing websocket-extensions 0.1.2
remote:        Installing mime-types-data 3.2016.0521
remote:        Installing arel 7.0.0
remote:        Installing coffee-script-source 1.10.0
remote:        Installing execjs 2.7.0
remote:        Installing method_source 0.8.2
remote:        Installing thor 0.19.1
remote:        Installing multi_json 1.12.1
```

```
remote:        安装带有本地扩展的 puma 3.4.0
remote:        安装带有本地扩展的 pg 0.18.4
remote:        使用 bundler 1.11.2
remote:        安装 sass 3.4.22
remote:        安装 tilt 2.0.5
remote:        安装 turbolinks-源 5.0.0
remote:        安装 tzinfo 1.2.2
remote:        安装带有本地扩展的 nokogiri 1.6.8
remote:        安装 rack-测试 0.6.3
remote:        安装 sprockets 3.6.3
remote:        安装带有本地扩展的 websocket-驱动 0.6.4
remote:        安装 mime-类型 3.1
remote:        安装 coffee-脚本 2.4.1
remote:        安装 uglifier 3.0.0
remote:        安装 turbolinks 5.0.0
remote:        安装 activesupport 5.0.0
remote:        安装 mail 2.6.4
remote:        安装 globalid 0.3.6
remote:        安装 activemodel 5.0.0
remote:        安装 jbuilder 2.5.0
remote:        安装 activejob 5.0.0
remote:        安装 activerecord 5.0.0
remote:        安装 loofah 2.0.3
remote:        安装 rails-dom-测试 2.0.1
remote:        安装 rails-html-清理器 1.0.3
remote:        安装 actionview 5.0.0
remote:        安装 actionpack 5.0.0
remote:        安装 actionmailer 5.0.0
remote:        安装 railties 5.0.0
remote:        安装 actioncable 5.0.0
remote:        安装 sprockets-rails 3.1.1
remote:        安装 coffee-rails 4.2.1
remote:        安装 jquery-rails 4.1.1
remote:        安装 rails 5.0.0
remote:        正在安装 sass-rails 5.0.5
remote:        Bundle 完成！15个 Gemfile 依赖，54个 gem 已安装。
remote:        组内开发和测试用的 Gems 未安装。
remote:        打包的 gems 安装在 ./vendor/bundle 目录中。
remote:        打包完成 (31.86秒)
remote:        正在清理 bundler 缓存。
remote:        警告：当前运行的 Bundler 版本低于创建锁文件时的版本。 建议您通过运行 `gem install bundler` 升级到最新
版本的 Bundler。
remote: -----> 正在为 Rails 资源管线准备应用
remote:        正在运行：rake assets:precompile
remote:        I, [2016-07-08T17:08:57.046245 #1222]  信息 -- : 正在写入
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-1bf5315c71171ad5f9cbef00193d56b
7e45263ddc64caf676ce988cfbb6570bd.js
remote:        I, [2016-07-08T17:08:57.046951 #1222]  信息 -- : 正在写入
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-1bf5315c71171ad5f9cbef00193d56b
7e45263ddc64caf676ce988cfbb6570bd.js.gz
remote:        I, [2016-07-08T17:08:57.060208 #1222]  信息 -- : 正在写入

e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css
remote:        I, [2016-07-08T17:08:57.060656 #1222]  信息 -- : 正在写入
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css.gz
remote:        资源预编译完成 (4.06秒)
remote:        正在清理资源
remote:        正在运行：rake assets:clean
remote:
remote: ###### 警告：
remote:        未检测到 Procfile，使用默认的 Web 服务器。
```



```
remote:        Installing puma 3.4.0 with native extensions
remote:        Installing pg 0.18.4 with native extensions
remote:        Using bundler 1.11.2
remote:        Installing sass 3.4.22
remote:        Installing tilt 2.0.5
remote:        Installing turbolinks-source 5.0.0
remote:        Installing tzinfo 1.2.2
remote:        Installing nokogiri 1.6.8 with native extensions
remote:        Installing rack-test 0.6.3
remote:        Installing sprockets 3.6.3
remote:        Installing websocket-driver 0.6.4 with native extensions
remote:        Installing mime-types 3.1
remote:        Installing coffee-script 2.4.1
remote:        Installing uglifier 3.0.0
remote:        Installing turbolinks 5.0.0
remote:        Installing activesupport 5.0.0
remote:        Installing mail 2.6.4
remote:        Installing globalid 0.3.6
remote:        Installing activemodel 5.0.0
remote:        Installing jbuilder 2.5.0
remote:        Installing activejob 5.0.0
remote:        Installing activerecord 5.0.0
remote:        Installing loofah 2.0.3
remote:        Installing rails-dom-testing 2.0.1
remote:        Installing rails-html-sanitizer 1.0.3
remote:        Installing actionview 5.0.0
remote:        Installing actionpack 5.0.0
remote:        Installing actionmailer 5.0.0
remote:        Installing railties 5.0.0
remote:        Installing actioncable 5.0.0
remote:        Installing sprockets-rails 3.1.1
remote:        Installing coffee-rails 4.2.1
remote:        Installing jquery-rails 4.1.1
remote:        Installing rails 5.0.0
remote:        Installing sass-rails 5.0.5
remote:        Bundle complete! 15 Gemfile dependencies, 54 gems now installed.
remote:        Gems in the groups development and test were not installed.
remote:        Bundled gems are installed into ./vendor/bundle.
remote:        Bundle completed (31.86s)
remote:        Cleaning up the bundler cache.
remote:        Warning: the running version of Bundler is older than the version that created the
lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install bundler`.
remote: -----> Preparing app for Rails asset pipeline
remote:        Running: rake assets:precompile
remote:        I, [2016-07-08T17:08:57.046245 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-1bf5315c71171ad5f9cbef00193d56b
7e45263ddc64caf676ce988cfbb6570bd.js
remote:        I, [2016-07-08T17:08:57.046951 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-1bf5315c71171ad5f9cbef00193d56b
7e45263ddc64caf676ce988cfbb6570bd.js.gz
remote:        I, [2016-07-08T17:08:57.060208 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css
remote:        I, [2016-07-08T17:08:57.060656 #1222]  INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855.css.gz
remote:        Asset precompilation completed (4.06s)
remote:        Cleaning assets
remote:        Running: rake assets:clean
remote:
remote: ###### WARNING:
remote:        No Procfile detected, using the default web server.
```

```
remote:        我们建议通过 Procfile 明确声明如何启动您的服务器进程。
remote:        https://devcenter.heroku.com/articles/ruby-default-web-server
remote:
remote: -----> 发现进程类型
remote:        Procfile 声明的类型      -> (无)
remote:        构建包的默认类型 -> 控制台, rake, web, worker
remote:
remote: -----> 正在压缩...
remote:        完成: 29.2M
remote: -----> 正在启动...
remote:        已发布 v5
remote:        https://example.herokuapp.com/ 已部署到 Heroku
remote:
remote: 验证部署... 完成。
To https://git.heroku.com/example.git
 * [new branch]      master -> master
```

如果您在应用程序中使用数据库，则需要通过运行以下命令手动迁移数据库：

```
$ heroku run rake db:migrate
```

在 heroku run 之后的任何命令都将在 Heroku dyno 上执行。您可以通过运行以下命令获得交互式 shell 会话：

```
$ heroku run bash
```

确保您有一个 dyno 正在运行 web 进程类型：

```
$ heroku ps:scale web=1
```

heroku ps 命令会列出您的应用程序正在运行的 dyno：

```
$ heroku ps
=== web (Standard-1X): bin/rails server -p $PORT -e $RAILS_ENV (1)
web.1: starting 2016/07/08 12:09:06 -0500 (~ 2s ago)
```

您现在可以通过浏览器访问应用，使用命令 heroku open。

```
$ heroku open
```

Heroku 会为你提供一个默认的网页 URL，位于 herokuapp.com 域名下。当你准备好进行生产环境扩展时，你可以添加你自己的自定义域名。

第37.2节：管理 Heroku 的生产和预发布环境

每个 Heroku 应用至少运行在两个环境中：Heroku 上（我们称之为生产环境）和你的本地机器（开发环境）。如果有多个人在开发该应用，那么你就有多个开发环境——通常每台机器一个。通常，每个开发者还会有一个测试环境来运行测试。不幸的是，随着环境差异变大，这种方法会失效。例如，Windows 和 Mac 提供的环境与 Heroku 上的 Linux 堆栈不同，因此你不能总是确定在本地开发环境中能正常工作的代码，部署到生产环境时也能以相同方式运行。

解决方案是拥有一个尽可能接近生产环境的预发布环境。这可以通过创建第二个 Heroku 应用来托管你的预发布应用实现。通过预发布环境，你可以在类似生产的环境中检查代码，


```
remote:        We recommend explicitly declaring how to boot your server process via a Procfile.
remote:        https://devcenter.heroku.com/articles/ruby-default-web-server
remote:
remote: -----> Discovering process types
remote:        Procfile declares types     -> (none)
remote:        Default types for buildpack -> console, rake, web, worker
remote:
remote: -----> Compressing...
remote:        Done: 29.2M
remote: -----> Launching...
remote:        Released v5
remote:        https://example.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/example.git
 * [new branch]      master -> master
```

If you are using the database in your application you need to manually migrate the database by running:

```
$ heroku run rake db:migrate
```

Any commands after `heroku run` will be executed on a Heroku dyno. You can obtain an interactive shell session by running:

```
$ heroku run bash
```

Ensure you have one dyno running the web process type:

```
$ heroku ps:scale web=1
```

The heroku ps command lists the running dynos of your application:

```
$ heroku ps
=== web (Standard-1X): bin/rails server -p $PORT -e $RAILS_ENV (1)
web.1: starting 2016/07/08 12:09:06 -0500 (~ 2s ago)
```

You can now visit the app in our browser with `heroku open`.

```
$ heroku open
```

Heroku gives you a default web URL in the `herokuapp.com` domain. When you are ready to scale up for production, you can add your own custom domain.

Section 37.2: Managing Production and staging environments for a Heroku

Every Heroku app runs in at least two environments: on Heroku (we'll call that production) and on your local machine (development). If more than one person is working on the app, then you've got multiple development environments - one per machine, usually. Usually, each developer will also have a test environment for running tests. Unfortunately, this approach breaks down as the environments become less similar. Windows and Macs, for instance, both provide different environments than the Linux stack on Heroku, so you can't always be sure that code that works in your local development environment will work the same way when you deploy it to production.

The solution is to have a staging environment that is as similar to production as is possible. This can be achieved by creating a second Heroku application that hosts your staging application. With staging, you can check your code in a

在影响实际用户之前。

从零开始

假设你有一个应用运行在本地机器上，并且你准备将其推送到 Heroku。我们需要创建两个远程环境，预发布和生产。为了养成先推送到预发布的习惯，我们将从以下命令开始：

```
$ heroku create --remote staging
创建 strong-river-216 ..... 完成
http://strong-river-216.heroku.com/ | https://git.heroku.com/strong-river-216.git
Git 远程仓库 staging 已添加
```

默认情况下，heroku CLI 会创建带有 heroku git 远程仓库的项目。这里，我们使用 --remote 标志指定了一个不同的名称，因此推送代码到 Heroku 以及针对应用运行命令的方式与通常的 git push heroku master 略有不同：

```
$ git push staging master
...
$ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb``
web.1: 已运行 21 秒
```

一旦你的 staging 应用正常启动并运行，你就可以创建生产环境应用：

```
$ heroku create --remote production
正在创建 fierce-ice-327 ..... 完成
http://fierce-ice-327.heroku.com/ | https://git.heroku.com/fierce-ice-327.git
Git 远程仓库 production 已添加
$ git push production master
...
$ heroku ps --remote production
=== web: `bundle exec puma -C config/puma.rb
web.1: 运行16秒
```

这样，你就拥有了两个独立的Heroku应用运行相同的代码库——一个是预发布环境，一个是生产环境，配置完全相同。只需记住，在日常工作中你必须指定要操作的应用。你可以使用'--remote'标志，或者通过git配置指定默认应用。

production-like setting before having it affect your actual users.

Starting from scratch

Assume you have an application running on your local machine, and you're ready to push it to Heroku. We'll need to create both remote environments, staging and production. To get in the habit of pushing to staging first, we'll start with this:

```
$ heroku create --remote staging
Creating strong-river-216.... done
http://strong-river-216.heroku.com/ | https://git.heroku.com/strong-river-216.git
Git remote staging added
```

By default, the heroku CLI creates projects with a heroku git remote. Here, we're specifying a different name with the --remote flag, so pushing code to Heroku and running commands against the app look a little different than the normal git push heroku master:

```
 $ git push staging master
...
 $ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb``
web.1: up for 21s
```

Once your staging app is up and running properly, you can create your production app:

```
$ heroku create --remote production
Creating fierce-ice-327.... done
http://fierce-ice-327.heroku.com/ | https://git.heroku.com/fierce-ice-327.git
Git remote production added
$ git push production master
...
$ heroku ps --remote production
=== web: `bundle exec puma -C config/puma.rb
web.1: up for 16s
```

And with that, you've got the same codebase running as two separate Heroku apps – one staging and one production, set up identically. Just remember you will have to specify which app you are going to operate on your daily work. You can either use flag '--remote' or use your git config to specify a default app.

第38章：ActiveSupport

第38.1节：核心扩展：字符串访问

String#at

返回字符串对象的子串。接口与String#[]相同。

```
str = "hello"
str.at(0)      # => "h"
str.at(1..3)   # => "ell"
str.at(-2)     # => "l"
str.at(-2..-1) # => "lo"
str.at(5)      # => nil
str.at(5..-1)  # => ""
```

String#from

返回从指定位置到字符串末尾的子串。

```
str = "hello"
str.from(0)  # => "hello"
str.from(3)  # => "lo"
str.from(-2) # => "lo"
```

String#to

返回从字符串开头到指定位置的子字符串。
如果位置为负数，则从字符串末尾开始计数。

```
str = "hello"
str.to(0)  # => "h"
str.to(3)  # => "hell"
str.to(-2) # => "hell"
```

from 和 to 可以配合使用。

```
str = "hello"
str.from(0).to(-1) # => "hello"
str.from(1).to(-2) # => "ell"
```

String#first

返回第一个字符，或指定数量的字符，最多不超过字符串长度。

```
str = "hello"
str.first    # => "h"
str.first(1) # => "h"
str.first(2) # => "he"
str.first(0) # => ""
str.first(6) # => "hello"
```

String#last

返回字符串的最后一个字符，或从字符串末尾向前数指定数量的字符。

```
str = "hello"
str.last    # => "o"
str.last(1) # => "o"
```


Chapter 38: ActiveSupport

Section 38.1: Core Extensions: String Access

String#at

Returns a substring of a string object. Same interface as `String#[]`.

```
str = "hello"
str.at(0)      # => "h"
str.at(1..3)   # => "ell"
str.at(-2)     # => "l"
str.at(-2..-1) # => "lo"
str.at(5)      # => nil
str.at(5..-1)  # => ""
```

String#from

Returns a substring from the given position to the end of the string.

```
str = "hello"
str.from(0)  # => "hello"
str.from(3)  # => "lo"
str.from(-2) # => "lo"
```

String#to

Returns a substring from the beginning of the string to the given position.
If the position is negative, it is counted from the end of the string.

```
str = "hello"
str.to(0)  # => "h"
str.to(3)  # => "hell"
str.to(-2) # => "hell"
```

`from` and `to` can be used in tandem.

```
str = "hello"
str.from(0).to(-1) # => "hello"
str.from(1).to(-2) # => "ell"
```

String#first

Returns the first character, or a given number of characters up to the length of the string.

```
str = "hello"
str.first    # => "h"
str.first(1) # => "h"
str.first(2) # => "he"
str.first(0) # => ""
str.first(6) # => "hello"
```

String#last

Returns the last character, or a given number of characters from the end of the string counting backwards.

```
str = "hello"
str.last    # => "o"
str.last(1) # => "o"
```

```
str.last(2) # => "lo"
str.last(0) # => ""
str.last(6) # => "hello"
```

第38.2节：核心扩展：字符串到日期/时间的转换

String#to_time

将字符串转换为时间值。参数form可以是:utc或:local，默认为:local。

```
"13-12-2012".to_time               # => 2012-12-13 00:00:00 +0100
"06:12".to_time                    # => 2012-12-13 06:12:00 +0100
"2012-12-13 06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time(:utc)   # => 2012-12-13 06:12:00 UTC
"12/13/2012".to_time               # => ArgumentError: argument out of range
```

String#to_date

将字符串转换为日期值。

```
"1-1-2012".to_date   # => Sun, 01 Jan 2012
"01/01/2012".to_date # => Sun, 01 Jan 2012
"2012-12-13".to_date # => Thu, 13 Dec 2012
"12/13/2012".to_date # => ArgumentError: invalid date
```

String#to_datetime

将字符串转换为日期时间值。

```
"1-1-2012".to_datetime             # => Sun, 01 Jan 2012 00:00:00 +0000
"01/01/2012 23:59:59".to_datetime # => Sun, 01 Jan 2012 23:59:59 +0000
"2012-12-13 12:50".to_datetime     # => Thu, 13 Dec 2012 12:50:00 +0000
"12/13/2012".to_datetime           # => ArgumentError: invalid date
```

第38.3节：核心扩展：字符串排除

String#exclude?

是String#include?的反义方法

```
"hello".exclude? "lo" # => false
"hello".exclude? "ol" # => true
"hello".exclude? ?h   # => false
```

第38.4节：核心扩展：字符串过滤器

String#squish

返回给定字符串的版本，去除首尾空白，并将内部所有连续的空白合并为单个空格。破坏性版本 squish! 直接作用于字符串实例。

处理 ASCII 和 Unicode 空白字符。

```
%{ 多行-字符串
    字符串 }.压缩                  # => "多行字符串"" foo    bar
 boo".压缩 # => "foo bar boo"
```



```
str.last(2) # => "lo"
str.last(0) # => ""
str.last(6) # => "hello"
```


Section 38.2: Core Extensions: String to Date/Time Conversion

String#to_time

Converts a string to a Time value. The `form` parameter can be either `:utc` or `:local`, defaults to `:local`.

```
"13-12-2012".to_time               # => 2012-12-13 00:00:00 +0100
"06:12".to_time                    # => 2012-12-13 06:12:00 +0100
"2012-12-13 06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time         # => 2012-12-13 06:12:00 +0100
"2012-12-13T06:12".to_time(:utc)   # => 2012-12-13 06:12:00 UTC
"12/13/2012".to_time               # => ArgumentError: argument out of range
```

String#to_date

Converts a string to a Date value.

```
"1-1-2012".to_date   # => Sun, 01 Jan 2012
"01/01/2012".to_date # => Sun, 01 Jan 2012
"2012-12-13".to_date # => Thu, 13 Dec 2012
"12/13/2012".to_date # => ArgumentError: invalid date
```

String#to_datetime

Converts a string to a DateTime value.

```
"1-1-2012".to_datetime             # => Sun, 01 Jan 2012 00:00:00 +0000
"01/01/2012 23:59:59".to_datetime # => Sun, 01 Jan 2012 23:59:59 +0000
"2012-12-13 12:50".to_datetime     # => Thu, 13 Dec 2012 12:50:00 +0000
"12/13/2012".to_datetime           # => ArgumentError: invalid date
```

Section 38.3: Core Extensions: String Exclusion

String#exclude?

The inverse of `String#include?`

```
"hello".exclude? "lo" # => false
"hello".exclude? "ol" # => true
"hello".exclude? ?h   # => false
```

Section 38.4: Core Extensions: String Filters

String#squish

Returns a version of the given string without leading or trailing whitespace, and combines all consecutive whitespace in the interior to single spaces. Destructive version `squish!` operates directly on the string instance.

Handles both ASCII and Unicode whitespace.

```
%{ Multi-line
   string }.squish                  # => "Multi-line string"
" foo   bar    \n   \t   boo".squish # => "foo bar boo"
```

String#remove

返回一个新字符串，移除所有匹配的模式。破坏性版本remove!直接作用于给定字符串。

```
str = "foo bar test"
str.remove(" test")                 # => "foo bar"
str.remove(" test", /bar/)          # => "foo "
```

String#truncate

如果字符串长度超过给定长度，返回一个在该长度截断的字符串副本。

```
“很久很久以前，在一个遥远的世界”.截断(27)
# => "很久很久以前，在一个遥远的世界..."
```

传入一个字符串或正则表达式 :separator 以在自然断点处截断

```
'很久很久以前，在一个遥远的世界'.truncate(27, separator: ' ')
# => "很久很久以前，在一个..."

'很久很久以前，在一个遥远的世界'.truncate(27, separator: /\s/)
# => "很久很久以前，在一个..."
```

String#truncate_words

返回一个在指定单词数后截断的字符串。

```
'很久很久以前，在一个遥远的世界'.truncate_words(4)
# => "很久很久以前，在一个..."
```

传入一个字符串或正则表达式以指定不同的单词分隔符

```
“曾经<br>在<br>一个<br>世界里”.truncate_words(5, 分隔符: '<br>')
# => "曾经<br>在<br>一个<br>世界里..."
```

最后的字符将被:omission字符串替换（默认为“...”）

```
“他们发现许多人睡得更好。”.truncate_words(5, 省略符: '...
（续）')
# => "他们发现许多人...（续）"
```

String#strip_heredoc

去除heredoc中的缩进。查找缩进最少的非空行，并移除该行开头的空白字符。

```
if options[:usage]
  puts <<-USAGE.strip_heredoc
    该命令执行此类和此类操作。

    支持的选项有：
      -h          此信息
      ...
  用法
结束
```

用户将看到

String#remove

Returns a new string with all occurrences of the patterns removed. Destructive version `remove!` operates directly on the given string.

```
str = "foo bar test"
str.remove(" test")                 # => "foo bar"
str.remove(" test", /bar/)          # => "foo "
```

String#truncate

Returns a copy of a given string truncated at a given length if the string is longer than the length.

```
'Once upon a time in a world far far away'.truncate(27)
# => "Once upon a time in a wo..."
```

Pass a string or regexp `:separator` to truncate at a natural break

```
'Once upon a time in a world far far away'.truncate(27, separator: ' ')
# => "Once upon a time in a..."

'Once upon a time in a world far far away'.truncate(27, separator: /\s/)
# => "Once upon a time in a..."
```

String#truncate_words

Returns a string truncated after a given number of words.

```
'Once upon a time in a world far far away'.truncate_words(4)
# => "Once upon a time..."
```

Pass a string or regexp to specify a different separator of words

```
'Once<br>upon<br>a<br>time<br>in<br>a<br>world'.truncate_words(5, separator: '<br>')
# => "Once<br>upon<br>a<br>time<br>in..."
```

The last characters will be replaced with the `:omission` string (defaults to "...")

```
'And they found that many people were sleeping better.'.truncate_words(5, omission: '...
(continued)')
# => "And they found that many... (continued)"
```

String#strip_heredoc

Strips indentation in heredocs. Looks for the least-indented non-empty line and removes that amount of leading whitespace.

```
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

NO_FORMAT]此命令执行某些操作。支持的选项有：-h 此信息 ...

第38.5节：核心扩展：字符串变形

String#pluralize

返回字符串的复数形式。可选地接受一个count参数，当count `== 1`时返回单数形式。
也接受一个locale参数，用于语言特定的复数化处理。

```
'post'.pluralize             # => "posts"
'octopus'.pluralize          # => "octopi"
'sheep'.pluralize            # => "sheep"
'words'.pluralize            # => "words"
'the blue mailman'.pluralize # => "the blue mailmen"
'CamelOctopus'.pluralize     # => "CamelOctopi"
'apple'.pluralize(1)         # => "apple"
'apple'.pluralize(2)         # => "apples"
'ley'.pluralize(:es)         # => "leyes"
'ley'.pluralize(1, :es)      # => "ley"
```

String#singularize

返回字符串的单数形式。接受一个可选的locale参数。

```
'posts'.singularize            # => "post"
'octopi'.singularize           # => "octopus"
'sheep'.singularize            # => "sheep"
'word'.singularize             # => "word"
'the blue mailmen'.singularize # => "the blue mailman"
'CamelOctopi'.singularize      # => "CamelOctopus"
'leyes'.singularize(:es)       # => "ley"
```

String#constantize

尝试查找字符串中指定名称的已声明常量。当名称不是驼峰命名法（CamelCase）或未初始化时，会抛出NameError异常。

```
'Module'.constantize  # => Module
'Class'.constantize   # => Class
'blargle'.constantize # => NameError: wrong constant name blargle
```

String#safe_constantize

执行constantize操作，但在抛出NameError异常时返回nil。

```
'Module'.safe_constantize  # => Module
'Class'.safe_constantize   # => Class
'blargle'.safe_constantize # => nil
```

String#camelize

默认将字符串转换为大驼峰式（UpperCamelCase），如果参数为:lower，则转换为小驼峰式（lowerCamelCase）。

别名：camelcase

注意：还会将/转换为::，这对于将路径转换为命名空间非常有用。

```
'active_record'.camelize                # => "ActiveRecord"
'active_record'.camelize(:lower)        # => "activeRecord"
'active_record/errors'.camelize         # => "ActiveRecord::Errors"
```


NO_FORMAT]This command does such and such. Supported options are: -h This message ...

Section 38.5: Core Extensions: String Inflection

String#pluralize

Returns of plural form of the string. Optionally takes a `count` parameter and returns singular form if `count == 1`. Also accepts a `locale` parameter for language-specific pluralization.

```
'post'.pluralize             # => "posts"
'octopus'.pluralize          # => "octopi"
'sheep'.pluralize            # => "sheep"
'words'.pluralize            # => "words"
'the blue mailman'.pluralize # => "the blue mailmen"
'CamelOctopus'.pluralize     # => "CamelOctopi"
'apple'.pluralize(1)         # => "apple"
'apple'.pluralize(2)         # => "apples"
'ley'.pluralize(:es)         # => "leyes"
'ley'.pluralize(1, :es)      # => "ley"
```

String#singularize

Returns the singular form of the string. Accepts an optional `locale` parameter.

```
'posts'.singularize            # => "post"
'octopi'.singularize           # => "octopus"
'sheep'.singularize            # => "sheep"
'word'.singularize             # => "word"
'the blue mailmen'.singularize # => "the blue mailman"
'CamelOctopi'.singularize      # => "CamelOctopus"
'leyes'.singularize(:es)       # => "ley"
```

String#constantize

Tries to find a declared constant with the name specified in the string. It raises a `NameError` when the name is not in CamelCase or is not initialized.

```
'Module'.constantize  # => Module
'Class'.constantize   # => Class
'blargle'.constantize # => NameError: wrong constant name blargle
```

String#safe_constantize

Performs a `constantize` but returns `nil` instead of raising `NameError`.

```
'Module'.safe_constantize  # => Module
'Class'.safe_constantize   # => Class
'blargle'.safe_constantize # => nil
```

String#camelize

Converts strings to UpperCamelCase by default, if `:lower` is given as param converts to lowerCamelCase instead.

alias: `camelcase`

Note: will also convert `/` to `::` which is useful for converting paths to namespaces.

```
'active_record'.camelize                # => "ActiveRecord"
'active_record'.camelize(:lower)        # => "activeRecord"
'active_record/errors'.camelize         # => "ActiveRecord::Errors"
```

```
'active_record/errors'.camelize(:lower) # => "activeRecord::Errors"
```

String#titleize

将所有单词首字母大写，并替换字符串中的某些字符，以创建更美观的标题。

别名：titlecase

```
'man from the boondocks'.titleize # => "Man From The Boondocks"
'x-men: the last stand'.titleize  # => "X Men: The Last Stand"
```

String#underscore

将字符串中的表达式转换为带下划线的小写形式。是camelize的逆操作。

注意：underscore还会将::替换为/，以将命名空间转换为路径。

```
'ActiveModel'.underscore         # => "active_model"
'ActiveModel::Errors'.underscore # => "active_model/errors"
```

String#dasherize

将字符串中的下划线替换为连字符。

```
'puni_puni'.dasherize # => "puni-puni"
```

String#demodulize

移除字符串中常量表达式的模块部分。

```
'ActiveRecord::CoreExtensions::String::Inflections'.demodulize # => "Inflections"
'Inflections'.demodulize                                       # => "Inflections"
'::Inflections'.demodulize                                     # => "Inflections"
''.demodulize                                                  # => ''
```

String#deconstantize

从字符串中的常量表达式中移除最右边的部分。

```
'Net::HTTP'.deconstantize   # => "Net"
'::Net::HTTP'.deconstantize # => "::Net"
'String'.deconstantize      # => ""
'::String'.deconstantize    # => ""
''.deconstantize            # => ""
```

String#parameterize

替换字符串中的特殊字符，使其可以用作“美观”URL的一部分。

```
"Donald E. Knuth".parameterize # => "donald-e-knuth"
```

使用:preserve_case参数保留字符串中字符的大小写。

```
"Donald E. Knuth".parameterize(preserve_case: true) # => "Donald-E-Knuth"
```

parameterize的一个非常常见的用例是重写ActiveRecord模型的to_param方法，以支持更具描述性的URL别名。

```
class Person < ActiveRecord::Base
  def to_param
    "#{id}-#{name.parameterize}"
```



```
'active_record/errors'.camelize(:lower) # => "activeRecord::Errors"
```

String#titleize

Capitalizes all the words and replaces some characters in the string to create a nicer looking title.

alias: `titlecase`

```
'man from the boondocks'.titleize # => "Man From The Boondocks"
'x-men: the last stand'.titleize  # => "X Men: The Last Stand"
```

String#underscore

Makes an underscored, lowercase form from the expression in the string. The reverse of `camelize`.

Note: `underscore` will also change `::` to `/` to convert namespaces to paths.

```
'ActiveModel'.underscore         # => "active_model"
'ActiveModel::Errors'.underscore # => "active_model/errors"
```

String#dasherize

Replaces underscores with dashes in the string.

```
'puni_puni'.dasherize # => "puni-puni"
```

String#demodulize

Removes the module part from the constant expression in the string.

```
'ActiveRecord::CoreExtensions::String::Inflections'.demodulize # => "Inflections"
'Inflections'.demodulize                                       # => "Inflections"
'::Inflections'.demodulize                                     # => "Inflections"
''.demodulize                                                  # => ''
```

String#deconstantize

Removes the rightmost segment from the constant expression in the string.

```
'Net::HTTP'.deconstantize   # => "Net"
'::Net::HTTP'.deconstantize # => "::Net"
'String'.deconstantize      # => ""
'::String'.deconstantize    # => ""
''.deconstantize            # => ""
```

String#parameterize

Replaces special characters in a string so that it may be used as part of a 'pretty' URL.

```
"Donald E. Knuth".parameterize # => "donald-e-knuth"
```

Preserve the case of the characters in a string with the `:preserve_case` argument.

```
"Donald E. Knuth".parameterize(preserve_case: true) # => "Donald-E-Knuth"
```

A very common use-case for `parameterize` is to override the `to_param` method of an ActiveRecord model to support more descriptive url slugs.

```
class Person < ActiveRecord::Base
  def to_param
    "#{id}-#{name.parameterize}"
```

```
  结束
结束

Person.find(1).to_param # => "1-donald-e-knuth"
```

String#tableize

创建一个类似Rails将模型名称转换为表名的表名。将字符串中的最后一个单词变为复数形式。

```
'RawScaledScorer'.tableize # => "raw_scaled_scorers"
'ham_and_egg'.tableize     # => "ham_and_eggs"
'fancyCategory'.tableize   # => "fancy_categories"
```

String#classify

从复数表名返回一个类名字符串，类似Rails将表名转换为模型名。

```
'ham_and_eggs'.classify # => "HamAndEgg"
'posts'.classify        # => "Post"
```

String#humanize

将首个单词首字母大写，将下划线替换为空格，并去除末尾的_id（如果存在）。

```
'employee_salary'.humanize              # => "员工薪水"
'author_id'.humanize                    # => "作者"
'author_id'.humanize(capitalize: false) # => "author"
'_id'.humanize                          # => "编号"
```

String#upcase_first

仅将第一个字符转换为大写。

```
'what a Lovely Day'.upcase_first # => "What a Lovely Day"
'w'.upcase_first                 # => "W"
''.upcase_first                  # => ""
```

String#foreign_key

从类名创建外键名称。传入false参数可禁用在名称和id之间添加下划线_。

```
'Message'.foreign_key        # => "message_id"
'Message'.foreign_key(false) # => "messageid"
'Admin::Post'.foreign_key    # => "post_id"
```



```
  end
end

Person.find(1).to_param # => "1-donald-e-knuth"
```

String#tableize

Creates the name of a table like Rails does for models to table names. Pluralizes the last word in the string.

```
'RawScaledScorer'.tableize # => "raw_scaled_scorers"
'ham_and_egg'.tableize     # => "ham_and_eggs"
'fancyCategory'.tableize   # => "fancy_categories"
```

String#classify

Returns a class name string from a plural table name like Rails does for table names to models.

```
'ham_and_eggs'.classify # => "HamAndEgg"
'posts'.classify        # => "Post"
```

String#humanize

Capitalizes the first word, turns underscores into spaces, and strips a trailing `_id` if present.

```
'employee_salary'.humanize              # => "Employee salary"
'author_id'.humanize                    # => "Author"
'author_id'.humanize(capitalize: false) # => "author"
'_id'.humanize                          # => "Id"
```

String#upcase_first

Converts just the first character to uppercase.

```
'what a Lovely Day'.upcase_first # => "What a Lovely Day"
'w'.upcase_first                 # => "W"
''.upcase_first                  # => ""
```

String#foreign_key

Creates a foreign key name from a class name. Pass **`false`** param to disable adding _ between name and `id`.

```
'Message'.foreign_key        # => "message_id"
'Message'.foreign_key(false) # => "messageid"
'Admin::Post'.foreign_key    # => "post_id"
```

第39章：表单助手

Rails 提供了用于生成表单标记的视图助手。

第39.1节：创建搜索表单

要创建搜索表单，请输入以下代码

```
<%= form_tag("/search", method: "get") do %>

  <%= text_field_tag(:q) %>
  <%= submit_tag("搜索") %>
<% end %>
```

- `form_tag`：这是创建表单的默认助手。它的第一个参数`/search`是动作，第二个参数指定HTTP方法。对于搜索表单，始终使用方法get非常重要。
- label_tag：此助手创建一个html <label> 标签。
- text_field_tag：这将创建一个类型为text的输入元素
- `submit_tag`：这将创建一个类型为submit的输入元素

第39.2节：下拉菜单

标准示例：

```
@models = Model.all
select_tag "models", options_from_collection_for_select(@models, "id", "name"), {}
```

这将生成以下HTML：David

最后一个参数是选项，接受以下内容：

```
{
multiple: false,
disabled: false,
include_blank: false,
prompt: false
}
```

更多示例可见：http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

第39.3节：表单元素的辅助方法

复选框

```
<%= check_box_tag(:pet_dog) %>
<%= label_tag(:pet_dog, "我有一只狗") %>
<%= check_box_tag(:pet_cat) %>
<%= label_tag(:pet_cat, "我有一只猫") %>
```

这将生成以下HTML

```
<input id="pet_dog" name="pet_dog" type="checkbox" value="1" />
<label for="pet_dog">我有一只狗</label>
<input id="pet_cat" name="pet_cat" type="checkbox" value="1" />
```


Chapter 39: Form Helpers

Rails provides view helpers for generating form markup.

Section 39.1: Creating a search form

To create a search form, enter the following code

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

- `form_tag`: This is the default helper for creating a form. It's first parameter, `/search` is the action and the second parameter specifies the HTTP method. For search forms, it is important to always use the method `get`
- `label_tag`: This helper creates an html **`<label>`** tag.
- `text_field_tag`: This will create an input element with type `text`
- `submit_tag`: This creates an input element with type `submit`

Section 39.2: Dropdown

Standard example:

```
@models = Model.all
select_tag "models", options_from_collection_for_select(@models, "id", "name"), {}
```

This will generate the following HTML: David

The last argument are options, which accepts the following:

```
{
multiple: false,
disabled: false,
include_blank: false,
prompt: false
}
```

More examples can be found: http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

Section 39.3: Helpers for form elements

Checkboxes

```
<%= 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 will generate the following html

```
<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">我有一只猫</label>
```

单选按钮

```
<%= radio_button_tag(:age, "child") %>
<%= label_tag(:age_child, "我未满18岁") %>
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "我已满18岁") %>
```

这将生成以下HTML

```
<input id="age_child" name="age" type="radio" value="child" />
<label for="age_child">我未满18岁</label>
<input id="age_adult" name="age" type="radio" value="adult" />
<label for="age_adult">我已满18岁</label>
```

文本区域

要创建更大的文本框，建议使用text_area_tag

```
<%= text_area_tag(:message, "这是一个较长的文本字段", size: "25x6") %>
```

这将生成以下HTML

```
<textarea id="message" name="message" cols="25" rows="6">这是一个较长的文本字段</textarea>
```

数字字段

这将创建一个input<type="number">元素

```
<%= number_field :product, :rating %>
```

要指定一个值的范围，我们可以使用in:选项

```
<%= number_field :product, :rating, in: 1..10 %>
```

密码字段

有时你希望用户输入的字符被掩码。这将生成一个<input type="password">

```
<%= password_field_tag(:password) %>
```

电子邮件字段

这将创建一个<input type="email">

```
<%= email_field(:user, :email) %>
```

电话字段

这将创建一个<input type="tel">。

```
<%= telephone_field :user, :phone %>
```



```
<label for="pet_cat">I own a cat</label>
```

Radio Buttons

```
<%= radio_button_tag(:age, "child") %>
<%= label_tag(:age_child, "I am younger than 18") %>
<%= radio_button_tag(:age, "adult") %>
<%= label_tag(:age_adult, "I'm over 18") %>
```

This generates the following HTML

```
<input id="age_child" name="age" type="radio" value="child" />
<label for="age_child">I am younger than 18</label>
<input id="age_adult" name="age" type="radio" value="adult" />
<label for="age_adult">I'm over 18</label>
```

Text Area

To create a larger text box, it is recommended to use the `text_area_tag`

```
<%= text_area_tag(:message, "This is a longer text field", size: "25x6") %>
```

This will create the following HTML

```
<textarea id="message" name="message" cols="25" rows="6">This is a longer text field</textarea>
```

Number Field

This will create an `input<type="number">` element

```
<%= number_field :product, :rating %>
```

To specify a range of values, we can use the `in:` option

```
<%= number_field :product, :rating, in: 1..10 %>
```

Password Field

Sometimes you want the characters typed by the user to be masked. This will generate an `<input type="password">`

```
<%= password_field_tag(:password) %>
```

Email Field

This will create an `<input type="email">`

```
<%= email_field(:user, :email) %>
```

Telephone Field

This will create an `<input type="tel">`.

```
<%= telephone_field :user, :phone %>
```

日期助手

- input[type="date"]

```
<%= date_field(:user, :reservation) %>
```

- input[type="week"]

```
<%= week_field(:user, :reservation) %>
```

- input[type="year"]

```
<%= year_field(:user, :reservation) %>
```

- input[type="time"]

```
<%= time_field(:user, :check_in) %>
```


Date Helpers

- input[type="date"]

```
<%= date_field(:user, :reservation) %>
```

- input[type="week"]

```
<%= week_field(:user, :reservation) %>
```

- input[type="year"]

```
<%= year_field(:user, :reservation) %>
```

- input[type="time"]

```
<%= time_field(:user, :check_in) %>
```

第40章：ActiveRecord事务

第40.1节：基本示例

例如：

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
mary.deposit(100)
end
```

只有在取款和存款都没有引发异常的情况下，此示例才会从大卫那里取钱并给玛丽。异常将导致回滚（ROLLBAC
K），将数据库恢复到事务开始之前的状态。
但请注意，对象的实例数据不会恢复到事务前的状态。

第40.2节：单个事务中的不同ActiveRecord类

虽然事务类方法是在某个ActiveRecord类上调用的，但事务块中的对象不必全部是该类的实例。这是因为事务是针对数据库连接的，而不是针对模型的。

在此示例中，尽管事务是在Account类上调用的，但余额记录是以事务方式保存的：

```
Account.transaction do
  balance.save!
account.save!
end
```

事务方法也可以作为模型实例方法使用。例如，你也可以这样做：

```
balance.transaction do
  balance.save!
account.save!
end
```

第40.3节：多个数据库连接

一个事务作用于单个数据库连接。如果你有多个特定于类的数据库，事务将无法保护它们之间的交互。一个解决方法是在你修改的每个类的模型上开始一个事务：

```
Student.transaction do
  Course.transaction do
    course.enroll(student)
    student.units += course.units
  end
end
```

这是一个糟糕的解决方案，但完全分布式事务超出了ActiveRecord的范围。

第40.4节：save和destroy自动包装在事务中

save和destroy都包装在一个事务中，确保你在验证或

Chapter 40: ActiveRecord Transactions

Section 40.1: Basic example

For example:

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

This example will only take money from David and give it to Mary if neither withdrawal nor deposit raise an exception. Exceptions will force a ROLLBACK that returns the database to the state before the transaction began. Be aware, though, that the objects will not have their instance data returned to their pre-transactional state.

Section 40.2: Different ActiveRecord classes in a single transaction

Though the transaction class method is called on some ActiveRecord class, the objects within the transaction block need not all be instances of that class. This is because transactions are per-database connection, not per-model.

In this example a balance record is transactionally saved even though transaction is called on the Account class:

```
Account.transaction do
  balance.save!
  account.save!
end
```

The transaction method is also available as a model instance method. For example, you can also do this:

```
balance.transaction do
  balance.save!
  account.save!
end
```

Section 40.3: Multiple database connections

A transaction acts on a single database connection. If you have multiple class-specific databases, the transaction will not protect interaction among them. One workaround is to begin a transaction on each class whose models you alter:

```
Student.transaction do
  Course.transaction do
    course.enroll(student)
    student.units += course.units
  end
end
```

This is a poor solution, but fully distributed transactions are beyond the scope of ActiveRecord.

Section 40.4: save and destroy are automatically wrapped in a transaction

Both #save and #destroy come wrapped in a transaction that ensures that whatever you do in validations or

回调中所做的任何操作都在其保护范围内执行。因此，你可以使用验证来检查事务依赖的值，或者你可以在回调中抛出异常以回滚，包括`after_*`回调。

因此，对数据库的更改在操作完成之前不会在你的连接之外被看到。例如，如果你尝试在`after_save`中更新搜索引擎的索引，索引器将看不到更新后的记录。只有`after_commit`回调会在更新提交后触发。

第40.5节：回调

与提交和回滚事务相关的回调有两种类型：`after_commit`和`after_rollback`。

after_commit回调在事务提交后立即对事务中保存或销毁的每条记录调用。after_rollback回调在事务或保存点回滚后立即对事务中保存或销毁的每条记录调用。

这些回调对于与其他系统交互非常有用，因为你可以确保回调只在数据库处于永久状态时执行。例如，after_commit是放置钩子的好位置，用于清除缓存，因为在事务内清除缓存可能会导致缓存在数据库更新之前被重新生成。

第40.6节：回滚事务

ActiveRecord::Base.transaction 使用 ActiveRecord::Rollback 异常来区分有意的回滚和其他异常情况。通常，抛出异常会导致 .transaction 方法回滚数据库事务并传递异常。但如果你抛出一个 ActiveRecord::Rollback 异常，那么数据库事务将被回滚，但异常不会被传递。

例如，你可以在控制器中这样做来回滚事务：

```
class BooksController < ActionController::Base
  def create
Book.transaction do
      book = Book.new(params[:book])
      book.save!
      if today_is_friday?
        # 系统必须在星期五失败，这样我们的支持部门
        # 才不会失业。我们静默地回滚这个事务
        # 而不告诉用户。
        raise ActiveRecord::Rollback, "Call tech support!"
      end
    end
    # ActiveRecord::Rollback 是唯一不会被
    # ActiveRecord::Base.transaction 传递的异常，
    # 所以即使是星期五，这行代码仍然会被执行。
    redirect_to root_url
  结束
结束
```


callbacks will happen under its protected cover. So you can use validations to check for values that the transaction depends on or you can raise exceptions in the callbacks to rollback, including `after_*` callbacks.

As a consequence changes to the database are not seen outside your connection until the operation is complete. For example, if you try to update the index of a search engine in `after_save` the indexer won't see the updated record. The `after_commit` callback is the only one that is triggered once the update is committed.

Section 40.5: Callbacks

There are two types of callbacks associated with committing and rolling back transactions: `after_commit` and `after_rollback`.

`after_commit` callbacks are called on every record saved or destroyed within a transaction immediately after the transaction is committed. `after_rollback` callbacks are called on every record saved or destroyed within a transaction immediately after the transaction or savepoint is rolled back.

These callbacks are useful for interacting with other systems since you will be guaranteed that the callback is only executed when the database is in a permanent state. For example, `after_commit` is a good spot to put in a hook to clearing a cache since clearing it from within a transaction could trigger the cache to be regenerated before the database is updated.

Section 40.6: Rolling back a transaction

`ActiveRecord::Base.transaction` uses the `ActiveRecord::Rollback` exception to distinguish a deliberate rollback from other exceptional situations. Normally, raising an exception will cause the `.transaction` method to rollback the database transaction and pass on the exception. But if you raise an `ActiveRecord::Rollback` exception, then the database transaction will be rolled back, without passing on the exception.

For example, you could do this in your controller to rollback a transaction:

```
class BooksController < ActionController::Base
  def create
    Book.transaction do
      book = Book.new(params[:book])
      book.save!
      if today_is_friday?
        # The system must fail on Friday so that our support department
        # won't be out of job. We silently rollback this transaction
        # without telling the user.
        raise ActiveRecord::Rollback, "Call tech support!"
      end
    end
    # ActiveRecord::Rollback is the only exception that won't be passed on
    # by ActiveRecord::Base.transaction, so this line will still be reached
    # even on Friday.
    redirect_to root_url
  end
end
```

第41章：RSpec与Ruby on Rails

第41.1节：安装RSpec

如果你想在Rails项目中使用RSpec，应该使用 rspec- rails gem，它可以自动为你生成辅助文件和spec文件（例如，当你使用 rails

generate创建模型、资源或脚手架时）。

在 Gemfile中的 :development和 :test组中添加 rspec- rails：

```
group  :development,  :test do
  gem 'rspec-rails', '~> 3.5'
end
```

运行 bundle来安装依赖。

使用以下命令初始化：

```
rails generate rspec:install
```

这将为你的测试创建一个 spec/文件夹，以及以下配置文件：

- . rspec包含命令行 rspec工具的默认选项spec/spec_helper.rb包含
- 基本的RSpec配置选项spec/rails_helper.rb添加了更具体用于RSpec
- 和Rails一起使用的配置选项。

所有这些文件都带有合理的默认设置，帮助你快速入门，但随着测试套件的增长，你可以根据需要添加功能和更改配置。

Chapter 41: RSpec and Ruby on Rails

Section 41.1: Installing RSpec

If you want to use RSpec for a Rails project, you should use the `rspec-rails` gem, which can generate helpers and spec files for you automatically (for example, when you create models, resources or scaffolds using `rails generate`).

Add `rspec-rails` to both the `:development` and `:test` groups in the `Gemfile`:

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Run `bundle` to install the dependencies.

Initialize it with:

```
rails generate rspec:install
```

This will create a `spec/` folder for your tests, along with the following configuration files:

- `.rspec` contains default options for the command-line `rspec` tool
- `spec/spec_helper.rb` includes basic RSpec configuration options
- `spec/rails_helper.rb` adds further configuration options that are more specific to use RSpec and Rails together.

All these files are written with sensible defaults to get you started, but you can add features and change configurations to suit your needs as your test suite grows.

第42章：装饰器模式

第42.1节：使用Draper装饰模型

Draper通过约定自动将模型与其装饰器匹配。

```
# app/decorators/user_decorator.rb
class UserDecorator < Draper::Decorator
  def full_name
    "#{object.first_name} #{object.last_name}"
  end

  def created_at
    Time.use_zone(h.current_user.timezone) do
      object.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  结束
结束
```

给定一个包含ActiveRecord对象的@user变量，你可以通过在@user上调用#decorate来访问装饰器，或者如果你想指定，可以直接调用Draper类。

```
<% user = @user.decorate %><!-- 或者 -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>加入时间：<%= user.created_at %></h3>
```

第42.2节：使用SimpleDelegator装饰模型

大多数Rails开发者一开始会直接在模板中修改他们的模型信息：

```
<h1><%= "#{ @user.first_name } #{ @user.last_name }" %></h1>
<h3>加入时间: <%= @user.created_at.in_time_zone(current_user.timezone).strftime("%A, %d %b %Y %l:%M
%p") %></h3>
```

对于数据量较大的模型，这种做法很快会变得繁琐，并导致逻辑在不同模板间复制粘贴。

这个示例使用了stdlib中的SimpleDelegator。

对SimpleDelegator对象的所有请求默认都会传递给父对象。你可以重写任何带有展示逻辑的方法，或者添加特定于该视图的新方法。

SimpleDelegator提供了两个方法：__setobj__用于设置被委托的对象，__getobj__用于获取该对象。

```
class UserDecorator < SimpleDelegator
  attr_reader :view
  def initialize(user, view)
    __setobj__ @user
    @view = view
  end

  # 新方法可以隐式调用父类的方法
  def full_name
    "#{ first_name } #{ last_name }"
```


Chapter 42: Decorator pattern

Section 42.1: Decorating a Model using Draper

Draper automatically matches up models with their decorators by convention.

```
# app/decorators/user_decorator.rb
class UserDecorator < Draper::Decorator
  def full_name
    "#{object.first_name} #{object.last_name}"
  end

  def created_at
    Time.use_zone(h.current_user.timezone) do
      object.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  end
end
```

Given a `@user` variable containing an ActiveRecord object, you can access your decorator by calling `#decorate` on the `@user`, or by specifying the Draper class if you want to be specific.

```
<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

Section 42.2: Decorating a Model using SimpleDelegator

Most Rails developers start by modifying their model information within the template itself:

```
<h1><%= "#{ @user.first_name } #{ @user.last_name }" %></h1>
<h3>joined: <%= @user.created_at.in_time_zone(current_user.timezone).strftime("%A, %d %b %Y %l:%M
%p") %></h3>
```

For models with a lot of data, this can quickly become cumbersome and lead to copy-pasting logic from one template to another.

This example uses `SimpleDelegator` from the stdlib.

All requests to a `SimpleDelegator` object are passed to the parent object by default. You can override any method with presentation logic, or you can add new methods that are specific to this view.

`SimpleDelegator` provides two methods: `__setobj__` to set what object is being delegated to, and `__getobj__` to get that object.

```
class UserDecorator < SimpleDelegator
  attr_reader :view
  def initialize(user, view)
    __setobj__ @user
    @view = view
  end

  # new methods can call methods on the parent implicitly
  def full_name
    "#{ first_name } #{ last_name }"
```

```
  end

  # 但是，如果你正在重写一个已有的方法，你需要
  # 使用 __getobj__
  def created_at
    Time.use_zone(view.current_user.timezone) do
      __getobj__.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  结束
结束
```

有些装饰器依赖魔法来连接这种行为，但你可以通过在页面上初始化对象，使展示逻辑的来源更加明显。

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>加入时间: <%= user.created_at %></h3>
```

通过将视图对象的引用传递给装饰器，我们仍然可以在构建展示逻辑时访问所有其他视图辅助方法，而无需包含它们。

现在视图模板只负责将数据插入页面，变得更加清晰。


```
  end

  # however, if you're overriding an existing method you need
  # to use __getobj__
  def created_at
    Time.use_zone(view.current_user.timezone) do
      __getobj__.created_at.strftime("%A, %d %b %Y %l:%M %p")
    end
  end
end
```

Some decorators rely on magic to wire-up this behavior, but you can make it more obvious where the presentation logic is coming from by initializing the object on the page.

```
<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>
```

By passing a reference to the view object into the decorator, we can still access all of the rest of the view helpers while building the presentation logic without having to include it.

Now the view template is only concerned with inserting data into the page, and it is much more clear.

第43章：Elasticsearch

第43.1节：Searchkick

如果你想快速设置elasticsearch，可以使用searchkick gem：

```
gem 'searchkick'
```

将searchkick添加到你想搜索的模型中。

```
class Product < ActiveRecord::Base
  searchkick
end
```

将数据添加到搜索索引中。

```
Product.reindex
```

查询时，使用：

```
products = Product.search "apples"
products.each do |product|
  puts product.name
end
```

非常快，不需要 Elasticsearch 知识 ;-)

更多信息请见：https://github.com/ankane/searchkick

第43.2节：安装与测试

本地开发首先要做的是在你的机器上安装 ElasticSearch 并测试它是否正在运行。它需要安装 Java。安装过程相当简单：

- Mac OS X: `brew install elasticsearch`
- Ubuntu: `sudo apt-get install elasticsearch`

然后启动它：

- Mac OS X: `brew services start elasticsearch`
- Ubuntu: `sudo service elasticsearch start`

测试它最简单的方法是使用 `curl`。启动可能需要几秒钟，所以如果一开始没有任何响应，不要惊慌。

```
curl localhost:9200
```

示例响应：

```
{
  "name" : "水人",
  "cluster_name" : "elasticsearch_gkbonetti",
  "version" : {
    "number" : "2.3.5",
    "build_hash" : "90f439ff60a3c0f497f91663701e64ccd01edbb4",
```


Chapter 43: Elasticsearch

Section 43.1: Searchkick

If you want to setup quickly elasticsearch you can use the searchkick gem :

```
gem 'searchkick'
```

Add searchkick to models you want to search.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Add data to the search index.

```
Product.reindex
```

And to query, use:

```
products = Product.search "apples"
products.each do |product|
  puts product.name
end
```

Pretty quick, elasticsearch knowledge not required ;-)

More information here : https://github.com/ankane/searchkick

Section 43.2: Installation and testing

The first thing you want to do for local development is install ElasticSearch in your machine and test it to see if it is running. It requires Java to be installed. The installation is pretty straightforward:

- Mac OS X: `brew install elasticsearch`
- Ubuntu: `sudo apt-get install elasticsearch`

Then start it:

- Mac OS X: `brew services start elasticsearch`
- Ubuntu: `sudo service elasticsearch start`

For testing it, the easiest way is with `curl`. It might take a few seconds for it to start, so don't panic if you don't get any response at first.

```
curl localhost:9200
```

Example response:

```
{
  "name" : "Hydro-Man",
  "cluster_name" : "elasticsearch_gkbonetti",
  "version" : {
    "number" : "2.3.5",
    "build_hash" : "90f439ff60a3c0f497f91663701e64ccd01edbb4",
```

```
    "build_timestamp" : "2016-07-27T10:36:52Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "你知道的，专为搜索而生"
}
```

第43.3节：设置开发工具

当你开始使用ElasticSearch（ES）时，拥有一个图形化工具来帮助你探索数据可能会很有用。一个名为elasticsearch-head的插件正是为此而设计。安装方法如下：

- 查找ES安装在哪个文件夹：`ls -l $(which elasticsearch)`
- cd进入该文件夹并运行插件安装程序：`elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- 在浏览器中打开 http://localhost:9200/_plugin/head/

如果一切按预期工作，你应该会看到一个漂亮的图形界面，可以在其中探索你的数据。

第43.4节：介绍

ElasticSearch 有一个文档完善的 JSON API，但你可能想使用一些库来帮你处理这些：

- Elasticsearch - 官方的 HTTP API 低级封装Elasticsearch-rails
- - 官方的高级 Rails 集成，帮助你使用 ActiveRecord 或 Repository 模式将 Rails 模型与 ElasticSearch 连接
- Chewy - 一个替代的非官方高级 Rails 集成，非常流行且文档 arguably 更好

我们使用第一个选项来测试连接：

```
gem install elasticsearch
```

然后启动 ruby 终端并试用：

```
require 'elasticsearch'

client = Elasticsearch::Client.new log: true
# 默认连接到 http://localhost:9200

client.transport.reload_connections!
client.cluster.health

client.search q: 'test'
```



```
    "build_timestamp" : "2016-07-27T10:36:52Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```

Section 43.3: Setting up tools for development

When you are getting started with ElasticSearch (ES) it might be good to have a graphical tool that helps you explore your data. A plugin called `elasticsearch-head` does just that. To install it, do the following:

- Find out in which folder ES is installed: `ls -l $(which elasticsearch)`
- `cd` into this folder and run the plugin installation binary: `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- Open `http://localhost:9200/_plugin/head/` in your browser

If everything worked as expected you should be seeing a nice GUI where you can explore your data.

Section 43.4: Introduction

ElasticSearch has a well-documented JSON API, but you'll probably want to use some libraries that handle that for you:

- `Elasticsearch` - the official low level wrapper for the HTTP API
- `Elasticsearch-rails` - the official high level Rails integration that helps you to connect your Rails models with ElasticSearch using either ActiveRecord or Repository pattern
- `Chewy` - An alternative, non-official high level Rails integration that is very popular and arguably has better documentation

Let's use the first option for testing the connection:

```
gem install elasticsearch
```

Then fire up the ruby terminal and try it out:

```
require 'elasticsearch'

client = Elasticsearch::Client.new log: true
# by default it connects to http://localhost:9200

client.transport.reload_connections!
client.cluster.health

client.search q: 'test'
```

第44章：使用 react-rails gem 在 Rails 中集成 React

第44.1节：使用 rails_react gem 在 Rails 中安装 React

将 react-rails 添加到你的 Gemfile：

```
gem 'react-rails'
```

然后安装：

```
bundle install
```

接下来，运行安装脚本：

```
rails g react:install
```

这将会：

创建一个 components.js 清单文件和一个 app/assets/javascripts/components/ 目录，你将在这里放置你的组件在 application.js 中放置以下内容：

```
//= require react
//= require react_ujs
//= require components
```

第44.2节：在您的应用程序中使用 react_rails

React.js 构建

您可以通过添加配置来选择在每个环境中提供哪种 React.js 构建（开发版、生产版，带或不带附加组件）。以下是默认设置：

```
# config/environments/development.rb
MyApp::Application.configure do
  config.react.variant = :development
end


# config/environments/production.rb
MyApp::Application.configure do
  config.react.variant = :production
end
```

要包含附加组件，请使用此配置：

```
MyApp::Application.configure do
  config.react.addons = true # 默认为 false
end
```

重启 Rails 服务器后，//= require react 将提供由配置指定的 React.js 构建版本。

Chapter 44: React with Rails using react-rails gem

Section 44.1: React installation for Rails using rails_react gem

Add react-rails to your Gemfile:

```
gem 'react-rails'
```

And install:

```
bundle install
```

Next, run the installation script:

```
rails g react:install
```

This will:

create a components.js manifest file and a app/assets/javascripts/components/ directory, where you will put your components place the following in your application.js:

```
//= require react
//= require react_ujs
//= require components
```

Section 44.2: Using react_rails within your application

React.js builds

You can pick which React.js build (development, production, with or without add-ons) to serve in each environment by adding a config. Here are the defaults:

```
# config/environments/development.rb
MyApp::Application.configure do
  config.react.variant = :development
end


# config/environments/production.rb
MyApp::Application.configure do
  config.react.variant = :production
end
```

To include add-ons, use this config:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

After restarting your Rails server, //= require react will provide the build of React.js which was specified by the configurations.

react-rails 提供了几个关于 React.js 版本和构建的其他选项。有关使用 react-source gem 或自行添加 React.js 副本的更多信息，请参见 VERSIONS.md。

JSX

安装 react-rails 后，重启服务器。现在，.js.jsx 文件将在资源管道中被转换。

BabelTransformer 选项

您可以使用 babel 的转换器和自定义插件，并通过添加以下

配置向 babel 转译器传递选项：

```
config.react.jsx_transform_options = {
blacklist: ['spec.functionName', 'validation.react', 'strict'], # 默认选项
  optional: ["transformerName"],  # 传递额外的 babel 选项
whitelist: ["useStrict"] # 更多选项[enter link description here][1]
}
```

在底层，react-rails 使用 ruby-babel-transpiler 进行转换。

第44.3节：渲染与挂载

react-rails 包含一个视图助手 (react_component) 和一个非侵入式 JavaScript 驱动（react_ujs），它们协同工作以将 React 组件放置在页面上。您应在清单文件中在 react 之后（如果使用 Turbolinks，则在 turbolinks 之后）引入 UJS 驱动。

视图助手会在页面上放置一个带有所请求组件类和属性的 div。例如：

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- 变成： -->
        <div data-react-class="HelloMessage" data-react-props="{&amp;quot;name&amp;quot;:&amp;quot;John&amp;quot;}"></div>
```

页面加载时，react_ujs 驱动程序会扫描页面并使用 data-react-class 和 data-react-props 挂载组件。

如果存在 Turbolinks，组件会在 page:change 事件时挂载，并在 page:before-unload 时卸载。推荐使用 Turbolinks >= 2.4.0，因为它提供了更好的事件支持。

在 Ajax 调用的情况下，可以通过调用 JavaScript 手动触发 UJS 挂载：

ReactRailsUJS.mountComponents() 视图助手的签名是：

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` 是一个字符串，表示一个全局可访问的组件类名。它可以包含点（例如，"MyApp.Header.MenuItem"）。

```
   `props` 是一个响应 `#to_json`  方法的对象，或者是一个已经字符串化的 JSON 对象
(例如，使用 Jbuilder 制作，见下方注释)。
```

html_options 可能包括：tag: 用于使用除 div 以外的元素来嵌入 data-react-class 和 data-react-props。prerender:true 表示在服务器端渲染组件。**其他 任何其他参数（例如 class:, id:）都会传递给 content_tag。

react-rails offers a few other options for versions & builds of React.js. See VERSIONS.md for more info about using the react-source gem or dropping in your own copies of React.js.

JSX

After installing react-rails, restart your server. Now, .js.jsx files will be transformed in the asset pipeline.

BabelTransformer options

You can use babel's transformers and custom plugins, and pass options to the babel transpiler adding following configurations:

```
config.react.jsx_transform_options = {
  blacklist: ['spec.functionName', 'validation.react', 'strict'], # default options
  optional: ["transformerName"],  # pass extra babel options
  whitelist: ["useStrict"] # even more options[enter link description here][1]
}
```

Under the hood, react-rails uses ruby-babel-transpiler, for transformation.

Section 44.3: Rendering & mounting

`react-rails` includes a view helper (`react_component`) and an unobtrusive JavaScript driver (react_ujs) which work together to put React components on the page. You should require the UJS driver in your manifest after react (and after turbolinks if you use Turbolinks).

The view helper puts a div on the page with the requested component class & props. For example:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-props="{&amp;quot;name&amp;quot;:&amp;quot;John&amp;quot;}"></div>
```

On page load, the react_ujs driver will scan the page and mount components using data-react-class and data-react-props.

If Turbolinks is present components are mounted on the page:change event, and unmounted on page:before-unload. Turbolinks >= 2.4.0 is recommended because it exposes better events.

In case of Ajax calls, the UJS mounting can be triggered manually by calling from javascript:

ReactRailsUJS.mountComponents() The view helper's signature is:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` is a string which names a globally-accessible component class. It may have dots (eg, "MyApp.Header.MenuItem").

```
   `props` is either an object that responds to `#to_json` or an    already-stringified JSON object
(eg, made with Jbuilder, see note    below).
```

`html_options` may include: `tag:` to use an element other than a div to embed data-react-class and data-react-props. `prerender: true` to render the component on the server. `**other` Any other arguments (eg class:, id:) are passed through to content_tag.

第45章：Rails 食谱 - 高级 Rails 配方/学习和编码技巧

第45.1节：使用 Rails 控制台操作表格

查看表格

```
ActiveRecord::Base.connection.tables
```

删除任意表格。

```
ActiveRecord::Base.connection.drop_table("users")
------------或者----------------------
ActiveRecord::Migration.drop_table(:users)
------------或者--------------------
ActiveRecord::Base.connection.execute("drop table users")
```

从现有列中移除索引

```
ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

其中country是迁移文件中的列名，users表中已添加索引，如下所示：

```
t.string :country,add_index: true
```

移除外键约束

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

其中menus与food_items存在一对多关系，并且各自有相应的迁移文件。

添加列

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

例如：

```
ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

第45.2节：Rails方法 - 返回布尔值

Rails模型中的任何方法都可以返回布尔值。

简单方法-

```
  ##该方法返回ActiveRecord::Relation
  def check_if_user_profile_is_complete
User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
  end
```


Chapter 45: Rails Cookbook - Advanced rails recipes/learnings and coding techniques

Section 45.1: Playing with Tables using rails console

View tables

```
ActiveRecord::Base.connection.tables
```

Delete any table.

```
   ActiveRecord::Base.connection.drop_table("users")
------------OR---------------------
ActiveRecord::Migration.drop_table(:users)
------------OR--------------------
ActiveRecord::Base.connection.execute("drop table users")
```

Remove index from existing column

```
ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

where `country` is a column name in the migration file with **already** added index in `users` table as shown below:

```
t.string :country,add_index: true
```

Remove foreign key constraint

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

where `menus has_many food_items` and their respective migrations too.

Add column

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

for example:

```
ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

Section 45.2: Rails methods - returning boolean values

Any method in Rails model can return boolean value.

simple method-

```
  ##this method return ActiveRecord::Relation
  def check_if_user_profile_is_complete
    User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
  end
```

再次是返回布尔值的简单方法-

```
  ##该方法返回布尔值（注意结果前的!!符号）
  def check_if_user_profile_is_complete
!!User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
  end
```

所以，同一个方法现在将返回布尔值，而不是其他任何值 :)。

第45.3节：处理错误 - 未定义的方法 `where` 针对 #<Array:0x000000071923f8>

有时我们想对返回的记录集合使用where查询，但该集合不是 ActiveRecord::Relation。因此会出现上述错误，因为Where子句是针对ActiveRecord而非Array的。

对此有一个明确的解决方案，即使用Joins。

示例：

> 假设我需要查找所有处于激活状态且不是id为10的用户(User)的用户资料(UserProfile)。

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).map(&:user).where.not(:id=>10
)
```

因此，上述查询在执行map后会失败，因为map返回的是一个array，无法与where子句配合使用。

但使用joins，则可以使其正常工作，

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).joins(:user).where.not(:id=>1
0)
```

由于joins将输出类似于map的记录，但它们将是ActiveRecord而不是一个Array。

Again simple method returning boolean value-

```
  ##this method return Boolean(NOTE THE !! signs before result)
  def check_if_user_profile_is_complete
    !!User.includes( :profile_pictures,:address,:contact_detail).where("user.id = ?",self)
  end
```

So,the same method will now return boolean instead of anything else :).

Section 45.3: Handling the error - undefined method `where' for #<Array:0x000000071923f8>

Sometimes we want to use a `where` query on a a collection of records returned which is not `ActiveRecord::Relation`.Hence we get the above error as `Where` clause is know to `ActiveRecord` and not to `Array`.

There is a precise solution for this by using `Joins`.

EXAMPLE:

> Suppose i need to find all user profiles(UserProfile) which are active which is not a user(User) with an id=10.

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).map(&:user).where.not(:id=>10
)
```

So above query will fail after `map` as `map` will return an `array` which will **not** work with `where` clause.

But using joins,will make it work,

```
UserProfiles.includes(:user=>:profile_pictures]).where(:active=>true).joins(:user).where.not(:id=>1
0)
```

As `joins` will output similar records like `map` but they will be `ActiveRecord` and **not** an `Array`.

第46章：多用途ActiveRecord列

第46.1节：保存对象

如果你有一个属性需要作为对象保存和从数据库中检索，那么使用serialize方法指定该属性的名称，它将被自动处理。

该属性必须声明为text字段。

在模型中你必须声明字段的类型（`Hash`或`Array`）

更多信息见：serialize >> apidock.com

第46.2节：操作方法

在你的迁移文件中

```
classUsers<ActiveRecord::Migration[5.0]
  defchange
create_table:usersdo|t|
      ...
       t.文本 :preference
t.文本 :tag
       ...
       t.时间戳
     结束
  结束
end
```

在你的模型中

```
类 用户 < ActiveRecord::Base
    序列化 :preferences, 哈希
    序列化 :tags, 数组
end
```


Chapter 46: Multipurpose ActiveRecord columns

Section 46.1: Saving an object

If you have an attribute that needs to be saved and retrieved to database as an object, then specify the name of that attribute using the `serialize` method and it will be handled automatically.

The attribute must be declared as a `text` field.

In the model you must declare the type of the field (`Hash` or `Array`)

More info at: serialize >> apidock.com

Section 46.2: How To

In your migration

```
class Users < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ...
      t.text :preference
      t.text :tag
      ...
      t.timestamps
    end
  end
end
```

In your model

```
class User < ActiveRecord::Base
    serialize :preferences, Hash
    serialize :tags, Array
end
```

第47章：类的组织

第47.1节：服务类

控制器是我们应用程序的入口点。然而，它不是唯一可能的入口点。我希望我的逻辑可以从以下位置访问：

- Rake任务
- 后台作业
- 控制台
- 测试

如果我把逻辑放到控制器里，它就无法从所有这些地方访问。所以我们尝试"瘦控制器，胖模型"的方法，把逻辑移到模型里。但移到哪个模型呢？如果某段逻辑涉及到用户（User）、购物车（Cart）和产品模型–应该放在哪里？

继承自`ActiveRecord::Base`的类已经承担了很多职责。它处理查询接口、关联和验证。如果你在模型中添加更多代码，它很快就会变成一个难以维护的混乱，拥有数百个公共方法。

服务只是一个普通的 Ruby 对象。它的类不必继承自任何特定的类。它的名称是一个动词短语，例如`CreateUserAccount`而不是`UserCreation`或`UserCreationService`。它位于 app/services 目录中。你需要自己创建这个目录，但 Rails 会为你自动加载其中的类。

服务对象只做一件事

服务对象（也称为方法对象）执行一个动作。它包含执行该动作的业务逻辑。以下是一个示例：

```
# app/services/accept_invite.rb
classAcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
UserMailer.invite_accepted(invite).deliver
  结束
结束
```

我遵循的三个约定是：

服务放在app/services目录下。我建议你为业务逻辑较重的领域使用子目录。例如：

- 文件`app/services/invite/accept.rb`将定义`Invite::Accept`，而`app/services/invite/create.rb`将定义`Invite::Create`
- 服务以动词开头（且不以Service结尾）：ApproveTransaction，SendTestNewsletter，`ImportUsersFromCsv`
- 服务响应call方法。我发现使用其他动词会显得有些多余：ApproveTransaction.approve()读起来不太顺畅。此外，call方法是lambda、procs和方法对象的事实标准方法。

优点

服务对象展示了我的应用程序的功能

Chapter 47: Class Organization

Section 47.1: Service Class

Controller is an entry point to our application. However, it's not the only possible entry point. I would like to have my logic accessible from:

- Rake tasks
- background jobs
- console
- tests

If I throw my logic into a controller it won't be accessible from all these places. So let's try "skinny controller, fat model" approach and move the logic to a model. But which one? If a given piece of logic involves `User`, `Cart` and `Product` models – where should it live?

A class which inherits from `ActiveRecord::Base` already has a lot of responsibilities. It handles query interface, associations and validations. If you add even more code to your model it will quickly become an unmaintainable mess with hundreds of public methods.

A service is just a regular Ruby object. Its class does not have to inherit from any specific class. Its name is a verb phrase, for example `CreateUserAccount` rather than `UserCreation` or `UserCreationService`. It lives in app/services directory. You have to create this directory by yourself, but Rails will autoload classes inside for you.

A service object does one thing

A service object (aka method object) performs one action. It holds the business logic to perform that action. Here is an example:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

The three conventions I follow are:

Services go under the `app/services directory`. I encourage you to use subdirectories for business logic-heavy domains. For instance:

- The file `app/services/invite/accept.rb` will define `Invite::Accept` while `app/services/invite/create.rb` will define `Invite::Create`
- Services start with a verb (and do not end with Service): `ApproveTransaction`, `SendTestNewsletter`, `ImportUsersFromCsv`
- Services respond to the `call` method. I found using another verb makes it a bit redundant: `ApproveTransaction.approve()` does not read well. Also, the `call` method is the de facto method for `lambda`, `procs`, and method objects.

Benefits

Service objects show what my application does

我只需浏览services目录就能了解我的应用程序做什么：`ApproveTransaction`，`CancelTransaction`，`BlockAccount`，`SendTransactionApprovalReminder`...

快速查看服务对象，我就知道涉及哪些业务逻辑。我不必通过controllers、ActiveRecord模型回调和观察者来理解“批准交易”涉及的内容。

清理模型和控制器

控制器将请求（参数、会话、Cookie）转换为参数，传递给服务，并根据服务响应进行重定向或渲染。

```
class InviteController < ApplicationController
 def accept
    invite = Invite.find_by_token!(params[:token])
    if AcceptInvite.call(invite, current_user)
      redirect_to invite.item, notice: "Welcome!"
    else
redirect_to '/', alert: "Oopsy!"
    结束
  结束
end
```

模型只处理关联、作用域、验证和持久化。

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
accepted_by_user_id: user.id,
     accepted_at: time
    )
  end
end
```

这使得模型和控制器更容易测试和维护！

何时使用服务类

当一个操作符合以下一个或多个条件时，使用服务对象：

- 该操作复杂（例如，在会计期间末结账）该操作涉及多个模型（例如，使用订单、信
- 用卡和客户对象进行电子商务购买）

- 该操作与外部服务交互（例如，发布到社交网络）该操作不是底层模型的核心关
- 注点（例如，在一定时间段后清理过时数据）。

- 执行该操作有多种方式（例如，使用访问令牌或密码进行身份验证）。

来源

亚当·涅杰尔斯基博客

酿酒厂博客

代码气候博客

I can just glance over the services directory to see what my application does: `ApproveTransaction`, `CancelTransaction`, `BlockAccount`, `SendTransactionApprovalReminder`...

A quick look into a service object and I know what business logic is involved. I don't have to go through the controllers, `ActiveRecord` model callbacks and observers to understand what "approving a transaction" involves.

Clean-up models and controllers

Controllers turn the request (params, session, cookies) into arguments, pass them down to the service and redirect or render according to the service response.

```
class InviteController < ApplicationController
 def accept
    invite = Invite.find_by_token!(params[:token])
    if AcceptInvite.call(invite, current_user)
      redirect_to invite.item, notice: "Welcome!"
    else
      redirect_to '/', alert: "Oopsy!"
    end
  end
end
```

Models only deal with associations, scopes, validations and persistence.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

This makes models and controllers much easier to test and maintain!

When to use Service Class

Reach for Service Objects when an action meets one or more of these criteria:

- The action is complex (e.g. closing the books at the end of an accounting period)
- The action reaches across multiple models (e.g. an e-commerce purchase using Order, CreditCard and Customer objects)
- The action interacts with an external service (e.g. posting to social networks)
- The action is not a core concern of the underlying model (e.g. sweeping up outdated data after a certain time period).
- There are multiple ways of performing the action (e.g. authenticating with an access token or password).

Sources

Adam Niedzielski Blog

Brew House Blog

Code Climate Blog

第47.2节：模型类别

```
class Post < ActiveRecord::Base
  belongs_to :user
 has_many :comments

  验证 :user，存在性：true
  验证 :title，存在性：true，长度：{ in: 6..40 }

  范围 :topic，-> (topic) { 连接(:topics).条件(topic: topic) }

  保存前 :update_slug
after_create:send_welcome_email

  def publish!
update(published_at: Time.now, published: true)
  end

  def self.find_by_slug(slug)
    find_by(slug: slug)
  end

  private

  def update_slug
    self.slug = title.join('-')
  end

  def send_welcome_email
    WelcomeMailer.welcome(self).deliver_now
  end
end
```

模型通常负责：

- 建立关系
- 验证数据
- 通过作用域和方法提供数据访问
- 执行与数据持久化相关的操作。

在最高层次上，模型描述领域概念并管理其持久化。

Section 47.2: Model Class

```
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :comments

  validates :user, presence: true
  validates :title, presence: true, length: { in: 6..40 }

  scope :topic, -> (topic) { joins(:topics).where(topic: topic) }

  before_save :update_slug
  after_create :send_welcome_email

  def publish!
    update(published_at: Time.now, published: true)
  end

  def self.find_by_slug(slug)
    find_by(slug: slug)
  end

  private

  def update_slug
    self.slug = title.join('-')
  end

  def send_welcome_email
    WelcomeMailer.welcome(self).deliver_now
  end
end
```

Models are typically responsible for:

- setting up relationships
- validating data
- providing access to data via scopes and methods
- Performing actions around persistence of data.

At the highest level, models describe domain concepts and manages their persistence.

第48章：浅层路由

第48.1节：浅层路由的使用

避免深层嵌套的一种方法（如上所建议）是生成作用于父级范围内的集合操作，以便了解层级结构，但不嵌套成员操作。换句话说，只构建带有最少信息以唯一标识资源的路由，如下所示：

```
resources :articles, shallow: true do
  resources :comments
resources :quotes
  resources :drafts
end
```

DSL 的 shallow 方法创建了一个作用域，在该作用域内所有嵌套都是浅层的。这会生成与前面示例相同的路由：

```
shallow do
resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

存在两个用于自定义浅层路由的作用域选项。:shallow_path 会在成员路径前加上指定的参数前缀：

```
scope shallow_path: "sekret" do
  resources :articles do
resources :comments, shallow: true
  end
end
```

使用 Rake 命令获取如下定义的生成路由：

```
rake routes
```


Chapter 48: Shallow Routing

Section 48.1: Use of shallow

One way to avoid deep nesting (as recommended above) is to generate the collection actions scoped under the parent, so as to get a sense of the hierarchy, but to not nest the member actions. In other words, to only build routes with the minimal amount of information to uniquely identify the resource, like this:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

The shallow method of the DSL creates a scope inside of which every nesting is shallow. This generates the same routes as the previous example:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

There exist two options for scope to customize shallow routes. :shallow_path prefixes member paths with the specified parameter:

```
scope shallow_path: "sekret" do
  resources :articles do
    resources :comments, shallow: true
  end
end
```

Use Rake Command for get generated routes as define below:

```
rake routes
```

第49章：模型状态：AASM

第49.1节：带有AASM的基本状态

通常你最终会创建包含状态的模型，并且该状态会在对象的生命周期内发生变化。

AASM 是一个有限状态机启用库，可以帮助你轻松地通过对象的流程设计。

在模型中使用这样的东西非常符合“胖模型，瘦控制器”的理念，这是Rails的最佳实践之一。模型是唯一负责管理其状态、状态变化以及由这些变化触发的事件的部分。

安装方法，在Gemfile中

```
gem 'aasm'
```

考虑一个应用，用户为产品报价。

```
class Quote

  include AASM

aasm do
state :requested, initial: true  # 用户看到产品并请求报价
    state :priced                # 卖家设定价格
状态 :已支付                  # 买方支付了价格
    状态 :已取消              # 买方不愿意支付价格
    状态 :已完成              # 产品已交付。

事件 :定价 执行
从状态: 请求中，转换到: :已定价
    结束

事件 :支付 执行
从状态: :已定价，转换到: :已支付，成功后: :设置支付日期
    结束

事件 :完成 执行
从状态: :已支付，转换到: :已完成，条件: 产品已交付？
    end

事件 :取消 执行
从状态: [:请求中，:已定价]，转换到: :已取消
        从状态: :已支付，转换到: 已取消，成功后: :撤销费用
    结束


  end

  private

  定义 设置支付日期
    更新 payed_at: Time.zone.now
  结束
end
```


Chapter 49: Model states: AASM

Section 49.1: Basic state with AASM

Usually you'll end up creating models which will contain a state, and that state will be changing during the lifespan of the object.

AASM is a finite state machine enabler library that can help you out with dealing with having an easy passing through the process design of your objects.

Having something like this in your model goes pretty aligned with the Fat Model, Skinny Controller idea, one of Rails best practices. The model is the sole responsible of managing its state, its changes and of generating the events triggered by those changes.

To install, in Gemfile

```
gem 'aasm'
```

Consider an App where the user Quotes a product for a price.

```
class Quote

  include AASM

  aasm do
    state :requested, initial: true  # User sees a product and requests a quote
    state :priced                    # Seller sets the price
    state :payed                     # Buyer pays the price
    state :canceled                  # The buyer is not willing to pay the price
    state :completed                 # The product has been delivered.

    event :price do
        transitions from: requested, to: :priced
    end

    event :pay do
        transitions from: :priced, to: :payed, success: :set_payment_date
    end

    event :complete do
        transitions from: :payed, to: :completed, guard: product_delivered?
    end

    event :cancel do
        transitions from: [:requested, :priced], to: :canceled
        transitions from: :payed, to: canceled, success: :reverse_charges
    end


  end

  private

  def set_payment_date
    update payed_at: Time.zone.now
  end
end
```

Quote 类的状态可以根据你的流程灵活设定。

你可以将状态视为过去的状态，就像之前的例子或算法中使用的时态，例如：定价、支付、交付等。状态的命名取决于你。从个人角度来看，过去时态的状态更合适，因为你的最终状态肯定是一个过去的动作，并且与事件名称更好地衔接，后面会进行解释。

注意：请注意你使用的名称，必须避免使用 Ruby 或 Ruby on Rails 的保留关键字，如 valid、end、being 等。

定义了状态和转换后，我们现在可以访问 AASM 创建的一些方法。

例如：

```
Quote.priced  # 显示所有具有 priced 事件的 Quote
quote.priced? # 表示该特定报价是否已定价
quote.price!  # 触发事件，将状态从 requested 转换为 priced。
```

如你所见，事件具有转换，这些转换决定了事件调用时状态的变化方式。如果由于当前状态事件无效，将会抛出错误。

事件和转换还有其他一些回调，例如

```
guard: product_delivered?
```

将调用 product_delivered? 方法，该方法返回布尔值。如果返回 false，则不会应用转换，且如果没有其他可用转换，状态将不会改变。

```
success: :reverse_charges
```

如果翻译成功，:reverse_charges 方法将被调用。

AASM 中还有其他带有更多回调的方法，但这将帮助你创建第一个具有有限状态的模型。

The Quote class' states can go however it's best for your process.

You can think of the states as being past, like in the previous example or algo in other tense, for example: pricing, paying, delivering, etc. The naming of the states depends on you. From a personal point a view, past states work better because your end state will surely be a past action and links up better with the event names, which will be explained later.

NOTE: Be careful what names you use, you have to worry about not using Ruby or Ruby on Rails reserved keywords, like `valid`, `end`, `being`, etc.

Having defined the states and transitions we can now access some methods created by AASM.

For example:

```
Quote.priced  # Shows all Quotes with priced events
quote.priced? # Indicates if that specific quote has been priced
quote.price!  # Triggers the event the would transition from requested to priced.
```

As you can see the event has transitions, this transitions determine the way the state will change upon the event call. If the event is invalid due to the current state an Error will be raised.

The events and transitions also have some other callbacks, for example

```
guard: product_delivered?
```

Will call the `product_delivered?` method which will return a boolean. If it turns out false, the transition will not be applied and if the no other transitions are available, the state won't change.

```
success: :reverse_charges
```

If that translation successfully happens the `:reverse_charges` method will be invoked.

There are several other methods in AASM with more callbacks in the process but this will help you creating your first models with finite states.

第50章：Rails 5 API 认证

第50.1节：使用 Rails 进行认证 authenticate_with_http_token

```
authenticate_with_http_token do |token, options|
  @user = User.find_by(auth_token: token)
end
```

你可以使用 curl 测试此端点，发送如下请求

```
curl -IH "Authorization: Token token=my-token" http://localhost:3000
```


Chapter 50: Rails 5 API Authetication

Section 50.1: Authentication with Rails authenticate_with_http_token

```
authenticate_with_http_token do |token, options|
  @user = User.find_by(auth_token: token)
end
```

You can test this endpoint with `curl` by making a request like

```
curl -IH "Authorization: Token token=my-token" http://localhost:3000
```

第51章：测试 Rails 应用程序

第51.1节：单元测试

单元测试是对应用程序的部分进行隔离测试。通常被测试的单元是一个类或模块。

```
let(:gift) { create :gift }

describe '#find' do
subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

source

这种测试尽可能直接且具体。

第51.2节：请求测试

请求测试是端到端测试，模拟用户的行为。

```
it '允许用户设置他们的偏好' do
  check 'Ruby'
点击 '保存并继续'
  期望(user.languages).to eq ['Ruby']
end
```

source

这种测试关注用户流程，运行系统的所有层，有时甚至渲染 javascript。

Chapter 51: Testing Rails Applications

Section 51.1: Unit Test

Unit tests test parts of the application in isolation. usually a unit under test is a class or module.

```
let(:gift) { create :gift }

describe '#find' do
  subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

source

This kind if test is as direct and specific as possible.

Section 51.2: Request Test

Request tests are end to end tests that imitate the behavior of a user.

```
it 'allows the user to set their preferences' do
  check 'Ruby'
  click_on 'Save and Continue'
  expect(user.languages).to eq ['Ruby']
end
```

source

This kind of test focuses on user flows and runs through all layers of the system sometimes even rendering javascript.

第52章：Active Jobs

第52.1节：介绍

自Rails 4.2起，Active Job是一个用于声明任务并使其在各种队列后端上运行的框架。适合使用Active Job的场景包括非阻塞且可以并行运行的定期或临时任务。

第52.2节：示例任务

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # 这将在稍后发生
    user.unsubscribe
  end
end
```

第52.3节：通过生成器创建Active Job

```
$ rails g job user_unsubscribe
```


Chapter 52: Active Jobs

Section 52.1: Introduction

Available since Rails 4.2, Active Job is a framework for declaring jobs and making them run on a variety of queuing backends. Recurring or punctual tasks that are not blocking and can be run in parallel are good use cases for Active Jobs.

Section 52.2: Sample Job

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # this will happen later
    user.unsubscribe
  end
end
```

Section 52.3: Creating an Active Job via the generator

```
$ rails g job user_unsubscribe
```

第53章：多年来的Rails框架

当你是Rails新手并且在处理遗留的Rails应用时，理解各个框架何时引入可能会令人困惑。本主题旨在成为跨Rails版本所有框架的权威列表。

第53.1节：如何查找当前Rails版本中可用的框架？

使用

```
config.frameworks
```

获取表示每个框架的Symbol数组的选项。

第53.2节：Rails 1.x中的Rails版本

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- Railties

第53.3节：Rails 2.x中的Rails框架

- ActionMailer
- ActionPack
- ActiveRecord
- *ActiveResource（ActiveWebService被ActiveResource取代，Rails默认从SOAP转向REST）*
- ActiveSupport
- Railties

第53.4节：Rails 3.x中的Rails框架

- ActionMailer
- ActionPack
- ActiveModel
- ActiveRecord
- ActiveResource
- ActiveSupport
- Railties

Chapter 53: Rails frameworks over the years

When you're new to Rails and working on legacy Rails applications, it can be confusing to understand which framework was introduced when. This topic is designed to be the *definitive* list of all frameworks across Rails versions.

Section 53.1: How to find what frameworks are available in the current version of Rails?

Use the

```
config.frameworks
```

option to get an array of `Symbol`s that represent each framework.

Section 53.2: Rails versions in Rails 1.x

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- Railties

Section 53.3: Rails frameworks in Rails 2.x

- ActionMailer
- ActionPack
- ActiveRecord
- ActiveResource (*ActiveWebService was replaced by ActiveResource, and with that, Rails moved from SOAP to REST by default*)
- ActiveSupport
- Railties

Section 53.4: Rails frameworks in Rails 3.x

- ActionMailer
- ActionPack
- ActiveModel
- ActiveRecord
- ActiveResource
- ActiveSupport
- Railties

第54章：Ruby on Rails中的嵌套表单

第54.1节：如何在Ruby on Rails中设置嵌套表单

首先需要有：一个包含has_many关联的模型与另一个模型。

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

在ProjectsController中：

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

在嵌套形式中，您可以同时创建带有父对象的子对象。

```
<%= nested_form_for @project do |f| %>
  <%= f.label :name %>
  <%= f.text_field :name %>

  <% # 现在是 `Todo` 对象的部分 %>
  <%= f.fields_for :todo do |todo_field| %>
    <%= todo_field.label :name %>
    <%= todo_field.text_field :name %>
  <% end %>
<% end %>
```

当我们使用Project.new初始化@project以创建一个新的Project对象时，创建Todo对象也必须有类似的东西，有多种方法可以实现：

1. 在Projectscontroller的new方法中，你可以写：@todo = @project.todos.build 或者 @todo = @project.todos.new 用于实例化一个新的Todo对象。
2. 您也可以在视图中这样做：<%= f.fields_for :todos, @project.todos.build %>

对于强参数，您可以通过以下方式包含它们：

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

由于Todo对象将通过创建Project对象来创建，因此你必须在通过在Project模型中添加以下行来指定这一点：

```
accepts_nested_attributes_for :todos
```


Chapter 54: Nested form in Ruby on Rails

Section 54.1: How to setup a nested form in Ruby on Rails

The first to thing to have: a model that contains a `has_many` relation with another model.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

In `ProjectsController`:

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

In a nested form, you can create child objects with a parent object at the same time.

```
<%= nested_form_for @project do |f| %>
  <%= f.label :name %>
  <%= f.text_field :name %>

  <% # Now comes the part for `Todo` object %>
  <%= f.fields_for :todo do |todo_field| %>
    <%= todo_field.label :name %>
    <%= todo_field.text_field :name %>
  <% end %>
<% end %>
```

As we initialized `@project` with `Project.new` to have something for creating a new `Project` object, same way for creating a `Todo` object, we have to have something like this, and there are multiple ways to do so:

1. In `Projectscontroller`, in `new` method, you can write: `@todo = @project.todos.build` or `@todo = @project.todos.new` to instantiate a new `Todo` object.
2. You can also do this in view: `<%= f.fields_for :todos, @project.todos.build %>`

For strong params, you can include them in the following way:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

Since, the `Todo` objects will be created through the creation of a `Project` object, so you have to specify this thing in `Project` model by adding the following line:

```
accepts_nested_attributes_for :todos
```

第55章：Factory Girl

第55.1节：定义工厂

如果你有一个带有name和email属性的ActiveRecord User类，你可以通过让FactoryGirl猜测来为它创建一个工厂：

```
FactoryGirl.define do
factory :user do # 它会猜测User类
    name      "John"
email     "john@example.com"
  结束
结束
```

或者你可以明确指定，甚至更改它的名称：

```
FactoryGirl.定义 do
  工厂 :user_jack, 类: User do
    名字     "Jack"
email     "jack@example.com"
  结束
结束
```

然后在你的规格中，你可以使用 FactoryGirl 的方法，如下所示：

```
# 创建一个未保存的 User 类实例，填充了 John 的数据
build(:user)
# 创建一个未保存的 User 类实例，填充了 Jack 的数据
build(:user_jack)
```

最常用的方法有：

```
# Build 返回一个未保存的实例
user = build(:user)

# Create 返回一个已保存的实例
user = create(:user)

# Attributes_for 返回用于构建实例的属性哈希
attrs = attributes_for(:user)
```


Chapter 55: Factory Girl

Section 55.1: Defining Factories

If you have a ActiveRecord User class with name and email attributes, you could create a factory for it by making the FactoryGirl guess it:

```
FactoryGirl.define do
  factory :user do # it will guess the User class
    name     "John"
    email    "john@example.com"
  end
end
```

Or you can make it explicit and even change its name:

```
FactoryGirl.define do
  factory :user_jack, class: User do
    name     "Jack"
    email    "jack@example.com"
  end
end
```

Then in your spec you can use the FactoryGirl's methods with these, like this:

```
# To create a non saved instance of the User class filled with John's data
build(:user)
# and to create a non saved instance of the User class filled with Jack's data
build(:user_jack)
```

The most common methods are:

```
# Build returns a non saved instance
user = build(:user)

# Create returns a saved instance
user = create(:user)

# Attributes_for returns a hash of the attributes used to build an instance
attrs = attributes_for(:user)
```

第56章：从特定文件夹导入整个CSV文件

在本例中，假设我们有许多产品CSV文件存放在一个文件夹中。每个CSV文件都需要通过控制台命令上传到我们的数据库。在新建或已有项目中运行以下命令以创建此模型。

第56.1节：通过控制台命令上传CSV

终端命令：

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

最新创建控制器。

终端命令：

```
rails g controller Products
```

控制器代码：

```
class HistoriesController < ApplicationController
    def create
file = Dir.glob("#{Rails.root}/public/products/**/*.csv") #=> 这是读取CSV文件的文件夹目录

file.each 做 |file|
            Product.导入(file)
        结束
    结束
结束
```

模型：

```
class Product< ApplicationRecord
  def self.import(file)
CSV.foreach(file.path, headers: true) do |row|
          Product.create! row.to_hash
      结束
  结束
end
```

routes.rb

```
resources :products
```

app/config/application.rb

```
require 'csv'
```

现在打开你的开发控制台并运行

```
=> ProductsController.new.create #=> 从你的文件夹目录上传整个CSV文件
```


Chapter 56: Import whole CSV files from specific folder

In this example, lets say we have many product CSV files in a folder. Each CSV file need to upload our database from our console write a command. Run the following command in a new or existing project to create this model.

Section 56.1: Uploads CSV from console command

Terminal Commands:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Lates create controller.

Terminal Commands:

```
rails g controller Products
```

Controller Code:

```
class HistoriesController < ApplicationController
    def create
        file = Dir.glob("#{Rails.root}/public/products/**/*.csv") #=> This folder directory where
read the CSV files
        file.each do |file|
            Product.import(file)
        end
    end
end
```

Model:

```
class Product< ApplicationRecord
  def self.import(file)
      CSV.foreach(file.path, headers: true) do |row|
          Product.create! row.to_hash
      end
  end
end
```

routes.rb

```
resources :products
```

app/config/application.rb

```
require 'csv'
```

Now open your development `console` & `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

第57章：Ruby on Rails代码的工具优化与清理

在开发大型Rails应用程序时，保持代码整洁有序即使对有经验的开发者来说也是一项挑战。幸运的是，有一整类的gem可以让这项工作变得轻松许多。

第57.1节：如果你想保持代码的可维护性、安全性和优化性，可以看看一些用于代码优化和清理的gem：

Bullet

这个gem让我印象深刻。bullet gem帮助你消除所有的N+1查询，以及不必要的预加载关联。一旦安装并开始在开发环境中访问各种路由，带有警告的弹窗会弹出，提示需要优化的数据库查询。它开箱即用，对于优化你的应用程序非常有帮助。

Rails最佳实践

用于查找Rails特定代码异味的静态代码分析器。它提供了多种建议；使用scope访问，限制自动生成的路由，添加数据库索引等。尽管如此，它包含了许多不错的建议，可以让你更好地理解如何重构代码并学习一些最佳实践。

Rubocop

一个Ruby静态代码分析器，你可以用它来检查代码是否符合Ruby社区的代码规范。该gem通过命令行报告风格违规，提供许多有用的代码重构建议，比如无用的变量赋值、插值中冗余使用Object#to_s，甚至未使用的方法参数。

好的一点是它高度可配置，因为如果你没有100%遵循Ruby风格指南（例如有很多尾随空格，或者即使不插值也使用双引号包裹字符串等），分析器可能会让人感到烦躁。

它分为4个子分析器（称为cops）：风格（Style）、语法检查（Lint）、度量（Metrics）和Rails。

Chapter 57: Tools for Ruby on Rails code optimization and cleanup

Keeping your code clean and organized while developing a large Rails application can be quite a challenge, even for an experienced developer. Fortunately, there is a whole category of gems that make this job much easier.

Section 57.1: If you want to keep your code maintainable, secure and optimized, look at some gems for code optimization and cleanup :

Bullet

This one particularly blew my mind. The bullet gem helps you kill all the N+1 queries, as well as unnecessarily eager loaded relations. Once you install it and start visiting various routes in development, alert boxes with warnings indicating database queries that need to be optimized will pop out. It works right out of the box and is extremely helpful for optimizing your application.

Rails Best Practices

Static code analyzer for finding Rails specific code smells. It offers a variety of suggestions; use scope access, restrict auto-generated routes, add database indexes, etc. Nevertheless, it contains lots of nice suggestions that will give you a better perspective on how to re-factor your code and learn some best practices.

Rubocop

A Ruby static code analyzer which you can use to check if your code complies with the Ruby community code guidelines. The gem reports style violations through the command line, with lots of useful code refactoring goodies such as useless variable assignment, redundant use of Object#to_s in interpolation or even unused method argument.

A good thing is that it's highly configurable, since the analyzer can be quite irritating if you're not following the Ruby style guide 100% (i.e. you have lots of trailing whitespaces or you double quote your strings even when not interpolating, etc.).

It's divided into 4 sub-analyzers (called cops): Style, Lint, Metrics and Rails.

第58章：ActiveJob

Active Job 是一个用于声明任务并使其在各种队列后端上运行的框架。这些任务可以是定期执行的清理工作、计费收费、邮件发送等。实际上，任何可以拆分成小工作单元并行运行的任务都可以。

第58.1节：创建任务

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # 稍后执行某些操作
  end
end
```

第58.2节：将任务入队

```
# 将任务入队，任务将在队列系统空闲时尽快执行。
GuestsCleanupJob.perform_later guest
```


Chapter 58: ActiveJob

Active Job is a framework for declaring jobs and making them run on a variety of queuing backends. These jobs can be everything from regularly scheduled clean-ups, to billing charges, to mailings. Anything that can be chopped up into small units of work and run in parallel, really.

Section 58.1: Create the Job

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Do something later
  end
end
```

Section 58.2: Enqueue the Job

```
# Enqueue a job to be performed as soon as the queuing system is free.
GuestsCleanupJob.perform_later guest
```

第59章：Active Model 序列化器

ActiveModelSerializers，简称 AMS，为你的 JSON 生成带来了“约定优于配置”的理念。
ActiveModelSerializers 通过两个组件工作：序列化器和适配器。序列化器描述应序列化哪些属性和关系。适配器描述属性和关系应如何序列化。

第59.1节：使用序列化器

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
attributes :body
end
```


Chapter 59: Active Model Serializers

ActiveModelSerializers, or AMS for short, bring 'convention over configuration' to your JSON generation.
ActiveModelSerializers work through two components: serializers and adapters. Serializers describe which attributes and relationships should be serialized. Adapters describe how attributes and relationships should be serialized.

Section 59.1: Using a serializer

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

第60章：Rails引擎 - 模块化Rails

Rails引擎快速概述

引擎是小型的Rails应用程序，可用于为承载它们的应用程序添加功能。
定义Ruby on Rails应用程序的类是`Rails::Application`，实际上它继承了大量行为自Rails::Engine，这是定义引擎的类。我们可以说，普通的Rails应用程序只是一个具有更多功能的引擎。

第60.1节：创建模块化应用

入门

首先，让我们生成一个新的Ruby on Rails应用程序：

```
rails new ModularTodo
```

下一步是生成一个引擎！

```
cd ModularTodo && rails plugin new todo --mountable
```

我们还将创建一个'engines'文件夹来存放引擎（即使我们只有一个！）。

```
mkdir engines && mv todo ./engines
```

引擎，就像 gems 一样，带有一个 gemspec 文件。让's 填写一些真实的值以避免警告。

```
 #ModularTodo/engines/todo/todo.gemspec
$:.push File.expand_path("../lib", __FILE__)

#维护你的 gem 版本：
require "todo/version"

#描述你的 gem 并声明其依赖：
Gem::Specification.new do |s|
s.name          = "todo"
s.version       = Todo::VERSION
s.authors       = ["蒂博·德尼泽"]
s.email         = ["bo@samurails.com"]
s.homepage      = "//samurails.com"
s.summary       = "待办模块"
s.description = "用于模块化Rails文章的待办模块"
s.license       = "MIT"

  #更多内容
  #...
end
```

现在我们需要将待办引擎添加到父应用的Gemfile中。

```
#ModularTodo/Gemfile
#其他 gems
gem 'todo'，路径：'engines/todo'
```


Chapter 60: Rails Engine - Modular Rails

Quick overview of Rails engines

Engines are small Rails applications that can be used to add functionalities to the application hosting them. The class defining a Ruby on Rails application is `Rails::Application` which actually inherits a lot of its behavior from `Rails::Engine`, the class defining an engine. We can say that a regular Rails application is simply an engine with more features.

Section 60.1: Create a modular app

Getting started

First, let's generate a new Ruby on Rails application:

```
rails new ModularTodo
```

The next step is to generate an engine!

```
cd ModularTodo && rails plugin new todo --mountable
```

We will also create an 'engines' folder to store the engines (even if we just have one!).

```
mkdir engines && mv todo ./engines
```

Engines, just like gems, come with a gemspec file. Let's put some real values to avoid warnings.

```
 #ModularTodo/engines/todo/todo.gemspec
$:.push File.expand_path("../lib", __FILE__)

#Maintain your gem's version:
require "todo/version"

#Describe your gem and declare its dependencies:
Gem::Specification.new do |s|
  s.name        = "todo"
  s.version     = Todo::VERSION
  s.authors     = ["Thibault Denizet"]
  s.email       = ["bo@samurails.com"]
  s.homepage    = "//samurails.com"
  s.summary     = "Todo Module"
  s.description = "Todo Module for Modular Rails article"
  s.license     = "MIT"

  #Moar stuff
  #...
end
```

Now we need to add the Todo engine to the parent application Gemfile.

```
#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'
```

让我们'运行 bundle install。你应该能在 gem 列表中看到以下内容：

```
使用 todo 0.0.1，来源于 engines/todo
```

太好了，我们的 Todo 引擎已正确加载！在开始编码之前，我们还有最后一件事要做：挂载 Todo 引擎。我们可以在父应用的 routes.rb 文件中完成这一步。

```
Rails.application.routes.draw do
  mount Todo::Engine => "/"，别名为：'todo'
end
```

我们将其挂载在 /，但也可以让它通过 /todo 访问。由于我们只有一个模块，/ 就足够了。

现在你可以启动服务器并在浏览器中查看。你应该会看到默认的 Rails 视图，因为我们还没有定义任何控制器/视图。让'我们现在来做这件事！

构建 Todo 列表

我们将在 Todo 模块内生成一个名为 Task 的模型，但为了正确迁移父应用的数据库，我们需要在 engine.rb 文件中添加一个小的初始化器。

```
#ModularTodo/engines/todo/lib/todo/engine.rb
module Todo
  class Engine < ::Rails::Engine
    isolate_namespace Todo

initializer :append_migrations do |app|
      unless app.root.to_s.match(root.to_s)
        config.paths["db/migrate"].expanded.each do |p|
          app.config.paths["db/migrate"] << p
        结束
      结束
    end

  结束
结束
```

就是这样，现在当我们从父应用运行迁移时，Todo引擎中的迁移也会被加载。

让我们创建Task模型。需要在引擎文件夹中运行scaffold命令。

```
cd engines/todo && rails g scaffold Task title:string content:text
```

从父文件夹运行迁移：

```
rake db:migrate
```

现在，我们只需要在Todo引擎内定义根路由：

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
root 'tasks#index'
end
```


Let's run `bundle install`. You should see the following in the list of gems:

```
Using todo 0.0.1 from source at engines/todo
```

Great, our Todo engine is loaded correctly! Before we start coding, we have one last thing to do: mount the Todo engine. We can do that in the routes.rb file in the parent app.

```
Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end
```

We are mounting it at `/` but we could also make it accessible at `/todo`. Since we have only one module, `/` is fine.

Now you can fire up your server and check it in your browser. You should see the default Rails view because we didn't define any controllers/views yet. Let's do that now!

Building the Todo list

We are going to scaffold a model named `Task` inside the Todo module but to correctly migrate the database from the parent application, we need to add a small initializer to the `engine.rb` file.

```
#ModularTodo/engines/todo/lib/todo/engine.rb
module Todo
  class Engine < ::Rails::Engine
    isolate_namespace Todo

    initializer :append_migrations do |app|
      unless app.root.to_s.match(root.to_s)
        config.paths["db/migrate"].expanded.each do |p|
          app.config.paths["db/migrate"] << p
        end
      end
    end

  end
end
```

That's it, now when we run migrations from the parent application, the migrations in the Todo engine will be loaded too.

Let's create the `Task` model. The `scaffold` command needs to be run from the engine folder.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Run the migrations from the parent folder:

```
rake db:migrate
```

Now, we just need to define the root route inside the Todo engine:

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end
```

你可以玩它，创建任务，删除它们... 哦等等，删除功能不起作用！为什么？！嗯，似乎 JQuery 没有加载，所以让我们把它添加到引擎内的 application.js 文件中！

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

太好了，现在我们可以删除任务了！

You can play with it, create tasks, delete them… Oh wait, the delete is not working! Why?! Well, it seems JQuery is not loaded, so let's add it to the `application.js` file inside the engine!

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Yay, now we can destroy tasks!

第61章：单表继承

单表继承（STI）是一种设计模式，其基于将多个继承自同一个基类模型的数据保存到数据库中的单个表中的想法。

第61.1节：基本示例

首先我们需要一个表来存储数据

```
class CreateUsers < ActiveRecord::Migration
  def change
create_table :users do |t|
t.字符串 :名称
t.string :password
t.string :type # <- 这使其成为 STI

t.timestamps
    结束
  结束
end
```

然后让我们创建一些模型

```
class User < ActiveRecord::Base
  validates_presence_of :password
  # 这是一个父类。所有共享的逻辑都写在这里
end

class Admin < User
  # 管理员必须拥有比普通用户更安全的密码
  # 我们可以在这里添加
validates :custom_password_validation
end

class Guest < User
  # 假设我们有一个访客类型的登录。
  # 它有一个不可更改的静态密码
validates_inclusion_of :password, in: ['guest_password']
end
```

当你执行 `Guest.create(name: 'Bob')` 时，ActiveRecord 会将其转换为在 Users 表中创建一条记录，其中 `type: 'Guest'`。

当你检索记录 `bob = User.where(name: 'Bob').first` 时，返回的对象将是 `Guest` 的一个实例，可以通过 `bob.becomes(User)` 强制将其视为 User。

becomes 在处理共享的局部视图或超类的路由/控制器而非子类时最为有用。

第61.2节：自定义继承列

默认情况下，STI 模型类名存储在名为 `type` 的列中。但可以通过在基类中重写 `inheritance_column` 的值来更改其名称。例如：

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # 也可以是字符串
```


Chapter 61: Single Table Inheritance

Single Table Inheritance (STI) is a design pattern which is based on the idea of saving the data of multiple models which are all inheriting from the same Base model, into a single table in the database.

Section 61.1: Basic example

First we need a table to hold our data

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :type # <- This makes it an STI

      t.timestamps
    end
  end
end
```

Then lets create some models

```
class User < ActiveRecord::Base
  validates_presence_of :password
  # This is a parent class. All shared logic goes here
end

class Admin < User
  # Admins must have more secure passwords than regular users
  # We can add it here
  validates :custom_password_validation
end

class Guest < User
  # Lets say that we have a guest type login.
  # It has a static password that cannot be changed
  validates_inclusion_of :password, in: ['guest_password']
end
```

When you do a `Guest.create(name: 'Bob')` ActiveRecord will translate this to create an entry in the Users table with `type: 'Guest'`.

When you retrieve the record `bob = User.where(name: 'Bob').first` the object returned will be an instance of `Guest`, which can be forcibly treated as a User with `bob.becomes(User)`

becomes is most useful when dealing with shared partials or routes/controllers of the superclass instead of the subclass.

Section 61.2: Custom inheritance column

By default STI model class name is stored in a column named `type`. But its name can be changed by overriding `inheritance_column` value in a base class. E.g.:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
```

```
end

class Admin < User; end
```

此情况下的迁移文件如下所示：

```
class CreateUsers < ActiveRecord::Migration
  def change
create_table :users do |t|
t.字符串 :名称
t.string :password
t.string :entity_type

t.timestamps
    结束
  结束
end
```

当你执行Admin.create时，该记录将以entity_type = "Admin"保存在users表中

第61.3节：带有type列且不使用STI的Rails模型

在Rails模型中拥有 type列但不调用STI，可以通过将:_type_disabled赋值给inheritance_column来实现：

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```



```
end

class Admin < User; end
```

Migration in this case will look as follows:

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :password
      t.string :entity_type

      t.timestamps
    end
  end
end
```

When you do `Admin.create`, this record will be saved in the users table with `entity_type = "Admin"`

Section 61.3: Rails model with type column and without STI

Having `type` column in a Rails model without invoking STI can be achieved by assigning `:_type_disabled` to `inheritance_column`:

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

第62章：ActiveRecord事务

ActiveRecord事务是保护性代码块，其中一系列ActiveRecord查询只有在全部成功作为一个原子操作时才会永久生效。

第62.1节：Active Record事务入门

Active Record事务既可以应用于模型类，也可以应用于模型实例，事务块中的对象不必全部是同一类的实例。这是因为事务是基于数据库连接的，而不是基于模型的。例如：

```
User.transaction do
  account.save!
profile.save!
  print "所有保存成功，返回1"
  return 1
end
rescue_from ActiveRecord::RecordInvalid do |exception|
  print "抛出异常，事务已回滚"
  render_error "failure", exception.record.errors.full_messages.to_sentence
end
```

使用带感叹号的 save 确保当抛出异常时事务会自动回滚，回滚后，控制权会转到异常的 rescue 块。 确保你在事务块中捕获 save! 抛出的异常！

如果你不想使用 save!，可以在保存失败时手动抛出 raise ActiveRecord::Rollback。你不需要处理这个异常。它会回滚事务并将控制权交给事务块之后的下一条语句。

```
User.transaction do
      if account.save && profile.save
        print "所有保存成功，返回 1"
        return 1
      else
        raise ActiveRecord::Rollback
      end
    end
    print "事务已回滚"
```


Chapter 62: ActiveRecord Transactions

ActiveRecord Transactions are protective blocks where sequence of active record queries are only permanent if they can all succeed as one atomic action.

Section 62.1: Getting Started with Active Record Transactions

Active Record Transactions can be applied to Model classes as well as Model instances, the objects within the transaction block need not all be instances of same class. This is because transactions are per-database connection, not per-model. For example:

```
User.transaction do
  account.save!
  profile.save!
  print "All saves success, returning 1"
  return 1
end
rescue_from ActiveRecord::RecordInvalid do |exception|
  print "Exception thrown, transaction rolledback"
  render_error "failure", exception.record.errors.full_messages.to_sentence
end
```

Using save with a bang ensures that transaction will be automatically rolled back when the exception is thrown and after the rollback, control goes to the rescue block for the exception. **Make sure you rescue the exceptions thrown from the save! in Transaction Block.**

If you don't want to use save!, you can manually raise `raise ActiveRecord::Rollback` when the save fails. You need not handle this exception. It will then rollback the transaction and take the control to the next statement after transaction block.

```
  User.transaction do
    if account.save && profile.save
      print "All saves success, returning 1"
      return 1
    else
      raise ActiveRecord::Rollback
    end
  end
  print "Transaction Rolled Back"
```

第63章：Turbolinks

Turbolinks 是一个 JavaScript 库，可以加快你网页应用的导航速度。当你点击链接时，Turbolinks 会自动获取页面，替换其 <body>，并合并其 <head>，所有这些操作都不会产生完整页面加载的开销。

第63.1节：绑定到 turbolink 的页面加载概念

使用 turbolinks 时，传统的用法：

```
$(document).ready(function() {
  // 很棒的代码
});
```

将不起作用。使用 turbolinks 时，$(document).ready() 事件只会触发一次：在初始页面加载时。从那时起，每当用户点击你网站上的链接时，turbolinks 会拦截链接点击事件并发起 ajax 请求以替换 <body> 标签并合并 <head> 标签。整个过程触发了 turbolinks 中的"访问"概念。因此，不要使用上面传统的 `document.ready()` 语法，你需要绑定 turbolinks 的访问事件，代码如下：

```
// 纯 JavaScript
document.addEventListener("turbolinks:load", function() {
  // 很棒的代码
});

// jQuery
$(document).on('turbolinks:load', function() {
  // 你的代码
});
```

第63.2节：在特定链接上禁用turbolinks

在特定链接上禁用 turbolinks 非常简单。根据官方 turbolinks 文档：

> 可以通过在链接或其任意祖先元素上添加 data-turbolinks="false" 来逐个链接禁用 turbolinks。

示例：

```
// 禁用此链接的 turbolinks
<a href="/" data-turbolinks="false">已禁用</a>

// 禁用 div 标签内所有嵌套链接的 turbolinks
<div data-turbolinks="false">
  <a href="/">我已被禁用</a>
  <a href="/">我也被禁用</a>
</div>

// 当祖先禁用 turbolinks 时，重新启用特定链接
<div data-turbolinks="false">
  <a href="/">我已被禁用</a>
  <a href="/" data-turbolinks="true">我已重新启用</a>
</div>
```


Chapter 63: Turbolinks

Turbolinks is a javascript library that makes navigating your web application faster. When you follow a link, Turbolinks automatically fetches the page, swaps in its <body>, and merges its <head>, all without incurring the cost of a full page load.

Section 63.1: Binding to turbolink's concept of a page load

With turbolinks, the traditional approach to using:

```
$(document).ready(function() {
  // awesome code
});
```

won't work. While using turbolinks, the `$(document).ready()` event will only fire once: on the initial page load. From that point on, whenever a user clicks a link on your website, turbolinks will intercept the link click event and make an ajax request to replace the <body> tag and to merge the <head> tags. The whole process triggers the notion of a "visit" in turbolinks land. Therefore, instead of using the traditional `document.ready()` syntax above, you'll have to bind to turbolink's visit event like so:

```
// pure js
document.addEventListener("turbolinks:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbolinks:load', function() {
  // your code
});
```

Section 63.2: Disable turbolinks on specific links

It is very easy to disable turbolinks on specific links. According to the official turbolinks documentation:

> Turbolinks can be disabled on a per-link basis by annotating a link or any of its ancestors with data-turbolinks="false".

Examples:

```
// disables turbolinks for this one link
<a href="/" data-turbolinks="false">Disabled</a>

// disables turbolinks for all links nested within the div tag
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/">I'm also disabled</a>
</div>

// re-enable specific link when ancestor has disabled turbolinks
<div data-turbolinks="false">
  <a href="/">I'm disabled</a>
  <a href="/" data-turbolinks="true">I'm re-enabled</a>
</div>
```

第63.3节：理解应用访问

应用访问是通过点击启用了 Turbolinks 的链接，或通过调用

```
Turbolinks.visit(location)
```

默认情况下，visit 函数使用“advance”操作。更易理解的是，visit 函数的默认行为是跳转到“location”参数指定的页面。每当访问一个页面时，turbolinks 会使用 history.pushState 向浏览器历史记录中添加一个新条目。历史记录很重要，因为 turbolinks 会尽可能利用历史记录从缓存加载页面。这使得频繁访问的页面能够实现极快的渲染速度。

但是，如果您想访问一个位置而不将任何历史记录推入堆栈，可以在 visit 函数中使用 'replace' 操作，方法如下：

```
// 使用链接
<a href="/edit" data-turbolinks-action="replace">编辑</a>

// 编程方式
Turbolinks.visit("/edit", { action: "replace" })
```

这将用新页面替换历史堆栈的顶部，使堆栈中的项目总数保持不变。

还有一个“restore”操作，帮助恢复访问，即用户点击浏览器的前进按钮或后退按钮时发生的访问。Turbolinks 内部处理这些类型的事件，并建议用户不要手动干预默认行为。

第63.4节：在访问开始前取消访问

Turbolinks 提供了一个事件监听器，可用于阻止访问发生。监听 turbolinks:before-visit 事件，以便在访问即将开始时收到通知。

在事件处理程序中，您可以使用：

```
// 纯 JavaScript
event.data.url
```

或者

```
// jQuery
$event.originalEvent.data.url
```

以获取访问的位置。然后可以通过调用以下方法取消访问：

```
event.preventDefault()
```

注意：

根据官方 turbolinks 文档：

> 恢复访问无法取消，且不会触发 turbolinks:before-visit 事件。

Section 63.3: Understanding Application Visits

Application visits are initiated by clicking a Turbolinks-enabled link, or programmatically by calling

```
Turbolinks.visit(location)
```

By default, the visit function uses the 'advance' action. More understandably, the default behavior for the visit function is to advance to the page indicated by the "location" parameter. Whenever a page is visited, turbolinks pushes a new entry onto the browser's history using `history.pushState`. The history is important because turbolinks will try to use the history to load pages from cache whenever possible. This allows for extremely fast page rendering for frequently visited pages.

However, if you want to visit a location without pushing any history onto the stack, you can use the 'replace' action on the visit function like so:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programmatically
Turbolinks.visit("/edit", { action: "replace" })
```

This will replace the top of the history stack with the new page so that the total number of items on the stack remains unchanged.

There is also a "restore" action that aids in restoration vists, the visits that occur as a result of the user clicking the forward button or back button on their browser. Turbolinks handles these types of events internally and recommends that users don't manually tamper with the default behavior.

Section 63.4: Cancelling visits before they begin

Turbolinks provides an event listener that can be used to stop visits from occurring. Listen to the `turbolinks:before-visit` event to be notified when a visit is about to commence.

In the event handler, you can use:

```
// pure javascript
event.data.url
```

or

```
// jQuery
$event.originalEvent.data.url
```

to retrieve the location of the visit. The visit can then be cancelled by calling:

```
event.preventDefault()
```

NOTE:

According to the official turbolinks docs:

> Restoration visits cannot be canceled and do not fire turbolinks:before-visit.

第 63.5 节：跨页面加载持久化元素

考虑以下情况：假设你是一个社交媒体网站的开发者，该网站允许用户与其他用户成为朋友，并使用 turbolinks 来加快页面加载速度。网站每个页面的右上角都有一个数字，显示用户当前拥有的朋友总数。假设你正在使用你的网站，并且你有 3 个朋友。每当添加一个新朋友时，你会运行一些 JavaScript来更新朋友计数器。假设你刚刚添加了一个新朋友，并且你的 JavaScript 正常运行，将页面右上角的朋友计数更新为 4。现在，假设你点击了浏览器的后退按钮。当页面加载时，你会注意到朋友计数器显示为 3，尽管你实际上有四个朋友。

这是一个相对常见的问题，turbolinks 为此提供了解决方案。问题出现的原因是因为当用户点击后退按钮时，turbolinks 会自动从缓存中加载页面。缓存的页面并不总是会与数据库保持同步更新。

为了解决这个问题，假设你在一个 id 为 "friend-count" 的 标签内渲染好友数量：

```
<div id="friend-count" data-turbolinks-permanent>3 个好友</div>
```

通过添加 data-turbolinks-permanent 属性，你告诉 turbolinks 在页面加载时保留某些元素。官方文档中说：

> 通过给元素指定 HTML id 并用 data-turbolinks-permanent 注释来指定永久元素。在每次渲染之前，Turbolinks 会通过 id 匹配所有永久元素，并将它们从原始页面转移到新页面，保留它们的数据和事件监听器。

Section 63.5: Persisting elements across page loads

Consider the following situation: Imagine that you are the developer of a social media website that allows users to be friends with other users and that employs turbolinks to make page loading faster. In the top right of every page on the site, there is a number indicating the total number of friends that a user currently has. Imagine you are using your site and that you have 3 friends. Whenever a new friend is added, you have some javascript that runs which updates the friend counter. Imagine that you just added a new friend and that your javascript ran properly and updated the friend count in the top right of the page to now render 4. Now, imagine that you click the browser's back button. When the page loads, you notice that the friend counter says 3 even though you have four friends.

This is a relatively common problem and one that turbolinks has provided a solution for. The reason the problem occurs is because turbolinks automatically loads pages from the cache when a user clicks the back button. The cached page won't always be updated with the database.

To solve this issue, imagine that you render the friend count inside a tag with an id of "friend-count":

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

By adding the `data-turbolinks-permanent` attribute, you're telling turbolinks to persist certain elements across page loads. The official docs say:

> Designate permanent elements by giving them an HTML id and annotating them with data-turbolinks-permanent. Before each render, Turbolinks matches all permanent elements by id and transfers them from the original page to the new page, preserving their data and event listeners.

第64章：友好 ID

FriendlyId 是 Active Record 中用于生成 slug 和永久链接插件的“瑞士军刀推土机”。它让你可以创建漂亮的 URL，并像使用数字 id 一样使用人类友好的字符串。使用 FriendlyId，可以轻松让你的
应用使用如下 URL：

http://example.com/states/washington

第64.1节：Rails 快速入门

```
rails new my_app
cd my_app
```

Gemfile

```
gem 'friendly_id', '~> 5.1.0' # 注意：Rails 4.0+ 必须使用 5.0.0 或更高版本
rails generate friendly_id
rails generate scaffold user name:string slug:string:uniq
rake db:migrate
```

编辑 app/models/user.rb

```
class User < ApplicationRecord
  extend FriendlyId
friendly_id :name, 使用: :slugged
end

User.create! name: "Joe Schmoe"

# 在你的控制器中将 User.find 改为 User.friendly.find
User.friendly.find(params[:id])
```

```
rails server
GET http://localhost:3000/users/joe-schmoe
```

```
# 如果你要在现有应用中添加 FriendlyId，并且需要
# 为现有用户生成 slug，可以从
# 控制台、runner 或添加 Rake 任务执行以下操作：
User.find_each(&:save)

查找器默认不再被重写。如果你想要使用友好查找，必须使用
Model.friendly.find而不是 Model.find。不过，你可以通过
使用:finders插件来恢复 FriendlyId 4版本风格的查找器


friendly_id :foo, 使用: :slugged # 你必须使用 MyClass.friendly.find('bar')
#或者...
friendly_id :foo, 使用: [:slugged, :finders] # 现在你可以直接使用 MyClass.find('bar')
```

新增了“candidates”功能，可以轻松设置一组备用 slug，用于唯一区分记录，而不是简单地追加序列号。例如：

```
class Restaurant < ActiveRecord::Base
  extend FriendlyId
friendly_id :slug_candidates, 使用: :slugged

  # 尝试基于以下字段按
  # 递增的特异性顺序构建 slug。
  def slug_candidates
```


Chapter 64: Friendly ID

FriendlyId is the "Swiss Army bulldozer" of slugging and permalink plugins for Active Record. It lets you create pretty URLs and work with human-friendly strings as if they were numeric ids. With FriendlyId, it's easy to make your application use URLs like:

http://example.com/states/washington

Section 64.1: Rails Quickstart

```
rails new my_app
cd my_app
```

Gemfile

```
gem 'friendly_id', '~> 5.1.0' # Note: You MUST use 5.0.0 or greater for Rails 4.0+
rails generate friendly_id
rails generate scaffold user name:string slug:string:uniq
rake db:migrate
```

edit app/models/user.rb

```
class User < ApplicationRecord
  extend FriendlyId
  friendly_id :name, use: :slugged
end

User.create! name: "Joe Schmoe"

# Change User.find to User.friendly.find in your controller
User.friendly.find(params[:id])
```

```
rails server
GET http://localhost:3000/users/joe-schmoe
```

```
# If you're adding FriendlyId to an existing app and need
# to generate slugs for existing users, do this from the
# console, runner, or add a Rake task:
User.find_each(&:save)

Finders are no longer overridden by default. If you want to do friendly finds, you must do
Model.friendly.find rather than Model.find. You can however restore FriendlyId 4-style finders by
using the :finders addon


friendly_id :foo, use: :slugged # you must do MyClass.friendly.find('bar')
#or...
friendly_id :foo, use: [:slugged, :finders] # you can now do MyClass.find('bar')
```

A new "candidates" functionality which makes it easy to set up a list of alternate slugs that can be used to uniquely distinguish records, rather than appending a sequence. For example:

```
class Restaurant < ActiveRecord::Base
  extend FriendlyId
  friendly_id :slug_candidates, use: :slugged

  # Try building a slug based on the following fields in
  # increasing order of specificity.
  def slug_candidates
```

```
    [
      :name,
      [:name, :city],
      [:name, :street, :city],
      [:name, :street_number, :street, :city]
    ]
  结束
结束
```

使用 friendly_id gem 设置 slug 长度限制？

```
def normalize_friendly_id(string)
   super[0..40]
end
```



```
    [
      :name,
      [:name, :city],
      [:name, :street, :city],
      [:name, :street_number, :street, :city]
    ]
  end
end
```

Set slug limit length using friendly_id gem?

```
def normalize_friendly_id(string)
   super[0..40]
end
```

第65章：安全存储认证密钥

许多第三方 API 需要密钥，以防止滥用。如果他们给你发放了密钥，非常重要的是不要将密钥提交到公共仓库，因为这会让其他人窃取你的密钥。

第65.1节：使用 Figaro 存储认证密钥

在你的 Gemfile 中添加 `gem 'figaro'` 并运行 `bundle install`。然后运行 `bundle exec figaro install`；这将创建 config/application.yml 并将其添加到你的 .gitignore 文件中，防止其被添加到版本控制中。

你可以以如下格式将密钥存储在 application.yml 中：

```
SECRET_NAME: secret_value
```

其中 SECRET_NAME 和 secret_value 是您的 API 密钥的名称和值。

您还需要在 config/secrets.yml 中为这些密钥命名。您可以在每个环境中使用不同的密钥。
该文件应如下所示：

```
development:
secret_name: <%= ENV["SECRET_NAME"] %>
test:
secret_name: <%= ENV["SECRET_NAME"] %>
production:
secret_name: <%= ENV["SECRET_NAME"] %>
```

这些密钥的使用方式各不相同，但例如开发环境中的 some_component 需要访问 secret_name。在 config/environments/development.rb 中，您可以这样写：

```
Rails.application.configure do
  config.some_component.configuration_hash = {
    :secret => Rails.application.secrets.secret_name
  }
end
```

最后，假设您想在 Heroku 上启动生产环境。此命令将把 config/environments/production.rb 中的值上传到 Heroku：

```
$ figaro heroku:set -e production
```


Chapter 65: Securely storing authentication keys

Many third-party APIs require a key, allowing them to prevent abuse. If they issue you a key, it's very important that you not commit the key into a public repository, as this will allow others to steal your key.

Section 65.1: Storing authentication keys with Figaro

Add `gem 'figaro'` to your Gemfile and run `bundle install`. Then run `bundle exec figaro install`; this will create config/application.yml and add it to your .gitignore file, preventing it from being added to version control.

You can store your keys in application.yml in this format:

```
SECRET_NAME: secret_value
```

where SECRET_NAME and secret_value are the name and value of your API key.

You also need to name these secrets in config/secrets.yml. You can have different secrets in each environment. The file should look like this:

```
development:
  secret_name: <%= ENV["SECRET_NAME"] %>
test:
  secret_name: <%= ENV["SECRET_NAME"] %>
production:
  secret_name: <%= ENV["SECRET_NAME"] %>
```

How you use these keys varies, but say for example `some_component` in the development environment needs access to `secret_name`. In config/environments/development.rb, you'd put:

```
Rails.application.configure do
  config.some_component.configuration_hash = {
    :secret => Rails.application.secrets.secret_name
  }
end
```

Finally, let's say you want to spin up a production environment on Heroku. This command will upload the values in config/environments/production.rb to Heroku:

```
$ figaro heroku:set -e production
```

第66章：使用Devise进行API认证

Devise是Rails的认证解决方案。在继续之前，我想先简单说明一下API。因此，API
不处理会话（是无状态的），这意味着它在你请求后提供响应，之后不需要
进一步关注，也就是说系统的工作不依赖于之前或未来的状态，因此每次
向服务器请求时都需要携带认证信息，并且应告知Devise不要存储
认证信息。

第66.1节：入门

首先我们将创建Rails项目并设置Devise

创建一个Rails应用

```
rails new devise_example
```

现在将Devise添加到gem列表中

> 你可以在Rails项目根目录找到名为“Gemfile”的文件

然后运行`bundle install`

接下来，你需要运行生成器：

```
rails generate devise:install
```

现在在控制台你会看到一些指示，按照提示操作即可。

生成 devise 模型

```
rails generate devise MODEL
```

然后运行 `rake db:migrate`

更多详情请访问：Devise Gem

认证令牌

认证令牌用于通过唯一令牌验证用户，因此在我们继续逻辑之前，
需要先向 Devise 模型添加 `auth_token` 字段

因此，

```
rails g migration add_authentication_token_to_users

class AddAuthenticationTokenToUsers < ActiveRecord::Migration
  def change
add_column :users, :auth_token, :string, default: ""
    add_index :users, :auth_token, unique: true
  结束
结束
```


Chapter 66: Authenticate Api using Devise

Devise is authentication solution for Rails. Before going any further i would like to add quick note on API. So API does not handle sessions (is stateless) which means one that provide response after you request, and then requires no further attention, which means no previous or future state is required for the system to work hence whenever we requesting to the server need to pass authentication details with all API and should tell Devise not to store authentication details.

Section 66.1: Getting Started

So first we will create rails project and setup device

create a rails application

```
rails new devise_example
```

now add devise to gem list

> you can find a file named 'Gemfile' at the root of rails project

Then run `bundle install`

Next, you need to run the generator:

```
rails generate devise:install
```

Now on console you can find few instructions just follow it.

Generate devise model

```
rails generate devise MODEL
```

Then run `rake db:migrate`

For more details go to: Devise Gem

Authentication Token

Authentication token is used to authenticate a user with a unique token, So Before we proceed with the logic first we need to add `auth_token` field to a Devise model

Hence,

```
rails g migration add_authentication_token_to_users

class AddAuthenticationTokenToUsers < ActiveRecord::Migration
  def change
    add_column :users, :auth_token, :string, default: ""
    add_index :users, :auth_token, unique: true
  end
end
```

然后运行 `rake db:migrate`

现在我们已经准备好使用auth_token进行身份验证

在`app/controllers/application_controllers.rb`中

首先添加这一行

```
respond_to :html, :json
```

这将帮助Rails应用同时响应html和json

然后

```
protect_from_forgery with: :null
```

将此处的:null更改，因为我们不处理会话。

现在我们将在application_controller中添加身份验证方法因此，默

认情况下Devise使用email作为唯一字段，我们也可以使用自定义字段，这里我们将使用user_email和auth_token进行身份验证。

```
before_filter do
user_email = params[:user_email].presence
    user       = user_email && User.find_by_email(user_email)

    if user && Devise.secure_compare(user.authentication_token, params[:auth_token])
      sign_in user, store: false
    end
  end
```

> 注意：以上代码完全基于您的逻辑，我只是试图解释工作示例

在上述代码的第6行，您可以看到我设置了store: false，这将防止在每个请求上创建会话，因此我们实现了无状态的请求

Then run `rake db:migrate`

Now we are all set to do authentication using `auth_token`

In `app/controllers/application_controllers.rb`

First this line to it

```
respond_to :html, :json
```

this will help rails application respond with both html and json

Then

```
protect_from_forgery with: :null
```

will change this `:null` as we are not dealing with sessions.

now we will add authentication method in application_controller

So, by default Devise uses email as unique field we can also use custom fields, for this case we will be authenticating using user_email and auth_token.

```
  before_filter do
    user_email = params[:user_email].presence
    user       = user_email && User.find_by_email(user_email)

    if user && Devise.secure_compare(user.authentication_token, params[:auth_token])
      sign_in user, store: false
    end
  end
```

> Note: Above code is purely based on your logic i am just trying to explain the working example

On line 6 in the above code you can see that i have set `store: false` which will prevent from creating a session on each requests hence we achieved stateless re

第67章：将React.js与Rails集成使用Hyperloop

本主题涵盖了使用Hyperloop gem将React.js与Rails集成

这里未涵盖的其他方法包括使用react-rails或react_on_rails gems。

第67.1节：向您的Rails应用添加一个简单的react组件（用ruby编写）

1.将hyperloop gem添加到您的rails（4.0 - 5.1）Gemfile中
2.bundle install
3.将hyperloop清单添加到application.js文件中：// app/assets/javascripts/application.js ... //= hyperloop-loader
4. 创建您的react组件，并将它们放置在hyperloop/components目录中# app/hyperloop/components/hello_world.rb class HelloWorld < Hyperloop::Component after_mount do every(1.second) { mutate.current_time(Time.now) } end render do "Hello World! 现在的时间是：#{state.current_time}" end end
5. 组件的行为就像视图。它们通过控制器中的render_component方法“挂载”：# 在控制器的某处：... def hello_world render_component # 根据方法名渲染 HelloWorld end

第67.2节：回调

```
# 所有 React 回调都支持使用类似 ActiveRecord 的语法

class SomeCallBacks < Hyperloop::Component
  before_mount do
    # 初始化内容 - 替代普通的类初始化方法
  end
after_mount do
    # 任何对实际生成的 DOM 节点或窗口行为的访问都写在这里
  end
before_unmount do
    # 任何清理操作（例如取消定时器等）
  end

  # 你也可以用常规方式指定一个方法：
before_mount :do_some_more_initialization
end
```

第67.3节：声明组件参数（props）

```
class Hello < Hyperloop::Component
  # 参数（即react的props）使用param宏声明
  param :guest
render do
    "Hello there #{params.guest}"
  结束
结束

# 以guest = "Matz"的方式“挂载”Hello组件
Hello(guest: 'Matz')
```


Chapter 67: Integrating React.js with Rails Using Hyperloop

This topic covers integrating React.js with Rails using the Hyperloop gem

Other approaches not covered here are using the react-rails or react_on_rails gems.

Section 67.1: Adding a simple react component (written in ruby) to your Rails app

1. Add the hyperloop gem to your rails (4.0 - 5.1) Gemfile
2. `bundle install`
3. Add the hyperloop manifest to the application.js file: // app/assets/javascripts/application.js ... //= hyperloop-loader
4. Create your react components, and place them in the `hyperloop/components` directory # app/hyperloop/components/hello_world.rb class HelloWorld < Hyperloop::Component after_mount do every(1.second) { mutate.current_time(Time.now) } end render do "Hello World! The time is now: #{state.current_time}" end end
5. Components act just like views. They are "mounted" using the `render_component` method in a controller: # somewhere in a controller: ... def hello_world render_component # renders HelloWorld based on method name end

Section 67.2: Callbacks

```
# all react callbacks are supported using active-record-like syntax

class SomeCallBacks < Hyperloop::Component
  before_mount do
    # initialize stuff - replaces normal class initialize method
  end
  after_mount do
    # any access to actual generated dom node, or window behaviors goes here
  end
  before_unmount do
    # any cleanups (i.e. cancel intervals etc)
  end

  # you can also specify a method the usual way:
  before_mount :do_some_more_initialization
end
```

Section 67.3: Declaring component parameters (props)

```
class Hello < Hyperloop::Component
  # params (= react props) are declared using the param macro
  param :guest
  render do
    "Hello there #{params.guest}"
  end
end

# to "mount" Hello with guest = "Matz" say
  Hello(guest: 'Matz')
```

```
# 参数可以赋予默认值：
参数 guest: 'friend' # 或者
  参数 :guest，默认值：'friend'
```

第67.4节：HTML标签

```
# HTML标签是内置的，且为大写
类 HTML示例 < Hyperloop::组件
  渲染 执行
DIV 执行
SPAN { "你好" }
      SPAN { "欢迎来到机器！" }
    结束
  结束
end
```

第67.5节：事件处理程序

```
# 事件处理程序通过'on'方法附加
类 点击我 < Hyperloop::组件
  渲染 执行
DIV 执行
SPAN { "你好" }
A { "点击我" }.on(:click) { alert('你做到了！' }
    结束
  结束
结束
```

第67.6节：状态

```
# 状态通过 'state' 方法读取，通过 'mutate' 更新
# 当状态改变时，会导致所有依赖的DOM元素重新渲染

class StateExample < Hyperloop::Component
  state count: 0  # 默认情况下状态初始化为nil
  render do
DIV 执行
SPAN { "你好" }
A { "点击我" }.on(:click) { mutate.count(state.count + 1) }
      DIV do
        "你已经点击了我 #{state.count} #{'次'.pluralize(state.count)}"
      end unless state.count == 0
    end
  结束
结束
```

注意，状态可以通过 Hyperloop::Stores 在组件之间共享


```
# params can be given a default value:
  param guest: 'friend' # or
  param :guest, default: 'friend'
```

Section 67.4: HTML Tags

```
# HTML tags are built in and are UPCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end
```

Section 67.5: Event Handlers

```
# Event handlers are attached using the 'on' method
class ClickMe < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { alert('you did it!' }
    end
  end
end
```

Section 67.6: States

```
# States are read using the 'state' method, and updated using 'mutate'
# when states change they cause re-render of all dependent dom elements

class StateExample < Hyperloop::Component
  state count: 0  # by default states are initialized to nil
  render do
    DIV do
      SPAN { "Hello There" }
      A { "Click Me" }.on(:click) { mutate.count(state.count + 1) }
      DIV do
        "You have clicked me #{state.count} #{'time'.pluralize(state.count)}"
      end unless state.count == 0
    end
  end
end
```

Note that states can be shared between components using Hyperloop::Stores

第68章：更改默认的Rails应用环境

本章将讨论如何更改环境，使得当有人输入 rails s 时，不是在开发环境启动，而是在他们想要的环境中启动。

第68.1节：在本地机器上运行

通常，当通过输入命令运行 Rails 环境时，这只是运行默认环境，通常是开发环境

```
rails s
```

可以使用标志-e选择特定环境，例如：

```
rails s -e test
```

这将运行测试环境。

默认环境可以通过编辑终端中的~/.bashrc文件并添加以下行来更改：

```
export RAILS_ENV=production in your
```

第68.2节：在服务器上运行

如果在使用Passenger的远程服务器上运行，请将apache.conf更改为您想使用的环境。
例如，在此情况下您会看到RailsEnv production。

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## 这是默认设置
</VirtualHost>
```


Chapter 68: Change a default Rails application enviornment

This will discuss how to change the environment so when someone types `rails s` they boot in not development but in the environment they want.

Section 68.1: Running on a local machine

Normally when rails environment is run by typing. This just runs the default environment which is usually `development`

```
rails s
```

The specific environment can be selected by using the flag `-e` for example:

```
rails s -e test
```

Which will run the test environment.

The default environment can be changed in terminal by editing the `~/.bashrc` file, and adding the following line:

```
export RAILS_ENV=production in your
```

Section 68.2: Running on a server

If running on a remote server that is using Passenger change apache.conf to to the environment you want to use. For example this case you see `RailsEnv production`.

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## This is the default
</VirtualHost>
```

第69章：Rails - 引擎

参数	目的
--mountable	选项告诉生成器你想创建一个“可挂载”的且命名空间隔离的引擎
--full	选项告诉生成器你想创建一个包含骨架结构的引擎

引擎可以被视为为其宿主应用提供功能的迷你应用。一个Rails
应用实际上只是一个“超级”引擎，Rails::Application类继承了大量行为
自Rails::Engine。

引擎是可重用的Rails应用/插件。它的工作方式类似于Gem。著名的引擎有Device、Spree等
可以轻松集成到Rails应用中。

第69.1节：著名的例子有

生成简单的博客引擎

```
rails plugin new [引擎名称] --可挂载的
```

著名的引擎示例有

Device（Rails的认证gem）

Spree（电子商务）

Chapter 69: Rails -Engines

Parameters	Purpose
--mountable	option tells the generator that you want to create a "mountable" and namespace-isolated engine
--full	option tells the generator that you want to create an engine, including a skeleton structure

Engines can be considered miniature applications that provide functionality to their host applications. A Rails application is actually just a "supercharged" engine, with the Rails::Application class inheriting a lot of its behavior from Rails::Engine.

Engines are the reusable rails applications/plugins. It works like a Gem. Famous engines are Device, Spree gems which can be integrated with rails applications easily.

Section 69.1: Famous examples are

Generating simple blog engine

```
rails plugin new [engine name] --mountable
```

Famous engines examples are

Device (authentication gem for rails)

Spree (Ecommerce)

第70章：向你的Rails应用添加Amazon RDS

创建AWS RDS实例并通过安装所需连接器配置database.yml文件的步骤。

第70.1节：假设我们正在将MYSQL RDS连接到你的Rails应用

创建MYSQL数据库的步骤

1.登录亚马逊账户并选择RDS服务
2. 从实例标签中选择启动数据库实例
3. 默认会选择MYSQL社区版，因此点击选择按钮
4. 选择数据库用途，比如生产环境，然后点击下一步
5. 提供mysql版本、存储大小、数据库实例标识符、主用户名和密码
   然后点击下一步
6. 输入数据库名称并点击启动数据库实例
7.请等待所有实例创建完成。实例创建完成后，您将看到一个端点，复制
   该入口点（即主机名）

安装连接器

将MySQL数据库适配器添加到您的项目的gemfile中，

```
gem 'mysql2'
```

安装您添加的gem，

```
bundle install
```

其他一些数据库适配器有，

- gem 'pg' 用于PostgreSQL
- gem 'activerecord-oracle_enhanced-adapter' 用于 Oracle
- gem 'sql_server' 用于 SQL Server

配置您的项目的 database.yml 文件 打开您的 config/database.yml 文件

```
production:
  adapter: mysql2
  encoding: utf8
database: <%= RDS_DB_NAME %>   # 您在创建数据库时输入的名称
  username: <%= RDS_USERNAME %> # 数据库主用户名
password: <%= RDS_PASSWORD %> # 数据库主密码
  host: <%= RDS_HOSTNAME %>      # 数据库实例入口
  port: <%= RDS_PORT %>          # 数据库端口。MySQL 默认为 3306
```


Chapter 70: Adding an Amazon RDS to your rails application

Steps to create an AWS RDS instance and configure your database.yml file by installing the required connectors.

Section 70.1: Consider we are connecting MYSQL RDS with your rails application

Steps to create MYSQL database

1. Login to amazon account and select RDS service
2. Select `Launch DB Instance` from the instance tab
3. By defaul MYSQL Community Edition will be selected, hence click the **SELECT** button
4. Select the database purpose, say `production` and click `next step`
5. Provide the `mysql version, storage size, DB Instance Identifier, Master Username and Password` and click `next step`
6. Enter `Database Name` and click `Launch DB Instance`
7. Please wait until all the instance gets created. Once the instance gets created you will find an Endpoint, copy this entry point (which is referred as hostname)

Installing connectors

Add the MySQL database adapter to your project's gemfile,

```
gem 'mysql2'
```

Install your added gems,

```
bundle install
```

Some other database adapters are,

- `gem 'pg'` for PostgreSQL
- `gem 'activerecord-oracle_enhanced-adapter'` for Oracle
- `gem 'sql_server'` for SQL Server

Configure your project's database.yml file Open your config/database.yml file

```
production:
  adapter: mysql2
  encoding: utf8
  database: <%= RDS_DB_NAME %>  # Which you have entered you creating database
  username: <%= RDS_USERNAME %> # db master username
  password: <%= RDS_PASSWORD %> # db master password
  host: <%= RDS_HOSTNAME %>     # db instance entrypoint
  port: <%= RDS_PORT %>         # db post. For MYSQL 3306
```

第71章：Rails中的支付功能

本文档旨在通过一个完整的示例，向您介绍如何使用 Ruby on Rails 实现不同的支付方式。

在示例中，我们将涵盖两个非常知名的支付平台：Stripe 和 Braintree。

第71.1节：如何集成 Stripe

将 Stripe gem 添加到我们的Gemfile

```
gem 'stripe'
```

添加初始化器/stripe.rb文件。该文件包含连接您的stripe账户所需的密钥。

```
require 'require_all'

Rails.configuration.stripe = {
    :publishable_key => ENV['STRIPE_PUBLISHABLE_KEY'],
    :secret_key      => ENV['STRIPE_SECRET_KEY']
}

Stripe.api_key = Rails.configuration.stripe[:secret_key]
```

如何在Stripe创建新客户

```
Stripe::Customer.create({email: email, source: payment_token})
```

此代码使用给定的电子邮件地址和来源在Stripe上创建一个新客户。

payment_token是客户端提供的令牌，包含信用卡或银行账户等支付方式。更多信息：Stripe.js客户端

如何从Stripe检索计划

```
Stripe::Plan.retrieve(stripe_plan_id)
```

此代码通过其ID从Stripe检索一个计划。

如何创建订阅

当我们有了客户和计划后，可以在Stripe上创建一个新的订阅。

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

它将创建一个新的订阅并向我们的用户收费。了解当我们为用户订阅计划时Stripe上实际发生的情况非常重要，更多信息请见：Stripe订阅生命周期。

如何对用户进行单次收费

有时我们只想对用户进行一次收费，为此我们将执行以下操作。

```
Stripe::Charge.create(amount:   amount, customer: customer, currency: currency)
```

在这种情况下，我们对用户收取一次指定金额的费用。

常见错误：

Chapter 71: Payment feature in rails

This document pretend to introduce you, with a complete example, how you can implement different payment methods with Ruby on Rails.

In the example, we will cover Stripe and Braintree two very well-known payment platforms.

Section 71.1: How to integrate with Stripe

Add Stripe gem to our `Gemfile`

```
gem 'stripe'
```

Add `initializers/stripe.rb` file. This file contains the necessary keys for connecting with your stripe account.

```
require 'require_all'

Rails.configuration.stripe = {
    :publishable_key => ENV['STRIPE_PUBLISHABLE_KEY'],
    :secret_key      => ENV['STRIPE_SECRET_KEY']
}

Stripe.api_key = Rails.configuration.stripe[:secret_key]
```

How to create a new customer to Stripe

```
Stripe::Customer.create({email: email, source: payment_token})
```

This code creates a new customer on Stripe with given email address and source.

`payment_token` is the token given from the client-side that contains a payment method like a credit card or bank account. More info: Stripe.js client-side

How to retrieve a plan from Stripe

```
Stripe::Plan.retrieve(stripe_plan_id)
```

This code retrieves a plan from Stripe by its id.

How to create a subscription

When we have a customer and a plan we can create a new subscription on Stripe.

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

It will create a new subscription and will charge our User. It's important to know what really happens on Stripe when we subscribe a user to a plan, you will find more info here: Stripe Subscription lifecycle.

How to charge a user with a single payment

Sometimes we want to charge our users just a single time, for do that we will do the next.

```
Stripe::Charge.create(amount:   amount, customer: customer, currency: currency)
```

In that case, we are charging our user one time for given amount.

Common errors:

- 金额必须以整数形式发送，也就是说，2000表示20个货币单位。请查看此示例
- 您不能对用户收取两种货币。如果用户过去任何时候曾以欧元（EUR）被收费，您不能再以美元（USD）向该用户收费。
- 您不能对没有来源（支付方式）的用户收费。

- The amount must be sent in integer form, that means, 2000 will be 20 units of currency. Check this example
- You cannot charge a user in two currencies. If the user was charged in EUR at any moment in the past you cannot charge the user in USD.
- You cannot charge user without source (payment method).

第72章：Docker上的Rails

本教程将以已安装Docker和一个Rails应用程序为起点

第72.1节：Docker和docker-compose

首先，我们需要创建我们的Dockerfile。一个很好的示例可以在Nick Janetakis的这个博客中找到。

这段代码包含将在我们的docker机器启动时执行的脚本。因此，
我们安装了所有必需的库，并以启动Puma（RoR开发服务器）结束

```
# 使用精简版的Ruby 2.3。
FROM ruby:2.3.0-slim

# 可选地设置维护者名称，让人们知道是谁制作了此镜像。
维护者 Nick Janetakis <nick.janetakis@gmail.com>

# 安装依赖：
# - build-essential：确保某些 gem 可以被编译
# - nodejs：编译资源文件
# - libpq-dev：通过 postgres gem 与 postgres 通信
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
      build-essential nodejs libpq-dev git


# 设置一个环境变量，用于存储应用在 Docker 镜像中的安装路径。
# 该名称仅为遵循惯例，与项目名称相同。
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# 这设置了命令运行的上下文目录，并在 Docker 官方网站有详细说明。

WORKDIR $INSTALL_PATH

# 我们希望 binstubs 可用，这样我们可以直接调用 sidekiq 以及
# 可能的其他二进制文件作为命令覆盖，而不依赖于
# bundle exec。
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# 将应用程序代码从当前目录的工作站复制到工作目录。

COPY . .

# 确保静态资源暴露到卷中，以便 nginx 之后可以读取
# 这些值。
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true


# 默认执行的命令是启动 Puma 服务器。
CMD bundle exec puma -C config/puma.rb
```


Chapter 72: Rails on docker

This tutorial will start with Docker installed and with a Rails app

Section 72.1: Docker and docker-compose

First of all, we will need to create our `Dockerfile`. A good example can be found on this blog by Nick Janetakis.

This code contains the script that will be executed on our docker machine at the moment of start.For this reason, we are installing all the required libraries and ends with the start of Puma (RoR dev server)

```
# Use the barebones version of Ruby 2.3.
FROM ruby:2.3.0-slim

# Optionally set a maintainer name to let people know who made this image.
MAINTAINER Nick Janetakis <nick.janetakis@gmail.com>

# Install dependencies:
# - build-essential: To ensure certain gems can be compiled
# - nodejs: Compile assets
# - libpq-dev: Communicate with postgres through the postgres gem
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
      build-essential nodejs libpq-dev git


# Set an environment variable to store where the app is installed to inside
# of the Docker image. The name matches the project name out of convention only.
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# This sets the context of where commands will be running in and is documented
# on Docker's website extensively.
WORKDIR $INSTALL_PATH

# We want binstubs to be available so we can directly call sidekiq and
# potentially other binaries as command overrides without depending on
# bundle exec.
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# Copy in the application code from your work station at the current directory
# over to the working directory.
COPY . .

# Ensure the static assets are exposed to a volume so that nginx can read
# in these values later.
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true


# The default command that gets run will be to start the Puma server.
CMD bundle exec puma -C config/puma.rb
```

此外，我们将使用 docker-compose，为此，我们将创建 docker-compose.yml。该文件的说明更像是一个 docker-compose 教程，而不是与 Rails 的集成，这里我将不做讲解。

```
version: '2'

services:
  backend:
    links:
      - #无论你需要链接什么，比如数据库
    build: .
    command: ./scripts/start.sh
    ports:
      - '3000:3000'
    volumes:
      -.:/backend
    volumes_from:
      -gembox
    env_file:
      -.dev-docker.env
    stdin_open: true
    tty: true
```

仅凭这两个文件，你就足够运行 docker-compose up并启动你的docker

Also, we will use docker-compose, for that, we will create `docker-compose.yml`. The explanation of this file will be more a docker-compose tutorial than an integration with Rails and I will not cover here.

```
version: '2'

services:
  backend:
    links:
      - #whatever you need to link like db
    build: .
    command: ./scripts/start.sh
    ports:
      - '3000:3000'
    volumes:
      - .:/backend
    volumes_from:
      - gembox
    env_file:
      - .dev-docker.env
    stdin_open: true
    tty: true
```

Just with these two files you will have enough to run `docker-compose up` and wake up your docker

附录A：保留字

你在使用这些词作为变量名、模型名、方法名等时应当小心。

A.1节：保留字列表

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- Array
- BasicSocket
- Benchmark
- Bignum
- Binding
- CGI
- CGIMethods
- 交叉编译
- 类
- 类可继承属性
- 可比较
- 条件变量
- 配置
- 续延
-
- DRbId转换
- DRb对象
- DRb未转储
- 数据
- 日期
- 日期时间
- 委托者
- 代理者
- 摘要
- 目录
-
- 文件结束错误
- ERB
- 可枚举
- 错误号
- 异常
- FALSE
- FalseClass
- Fcntl
- 文件
- 文件列表
- 文件任务

Appendix A: Reserved Words

You should be careful using these words for variable, model name, method name or etc.

Section A.1: Reserved Word List

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- Array
- BasicSocket
- Benchmark
- Bignum
- Binding
- CGI
- CGIMethods
- CROSS_COMPILING
- Class
- ClassInheritableAttributes
- Comparable
- ConditionVariable
- Config
- Continuation
- DRb
- DRbIdConv
- DRbObject
- DRbUndumped
- Data
- Date
- DateTime
- Delegater
- Delegator
- Digest
- Dir
- ENV
- EOFError
- ERB
- Enumerable
- Errno
- Exception
- FALSE
- FalseClass
- Fcntl
- File
- FileList
- FileTask

- 文件测试
- 文件工具
- 整数
- 浮点数
- 浮点域错误
- GC
- 宝石
- 长选项解析
- 哈希
- 输入输出
- IO错误
- IPSocket
- IPsocket
- 索引错误(IndexError)
- 变形器(Inflector)
- 整数(Integer)
- 中断(Interrupt)
- 内核(Kernel)
- LN_SUPPORTED
- 加载错误(LoadError)
- 局部跳转错误(LocalJumpError)
- 日志记录器(Logger)
- 序列化(Marshal)
- 匹配数据(MatchData)
- 匹配数据(MatchingData)
- 数学(Math)
- 方法(Method)
- 模块(Module)
- 互斥锁(Mutex)
- MySQL
- MySQL错误(MysqlError)
- MySQL字段(MysqlField)
- MySQL结果(MysqlRes)
- 空值(NIL)
- 名称错误(NameError)
- 空类(NilClass)
- NoMemoryError
- NoMethodError
- NoWrite
- NotImplementedError
- Numeric
- OPT_TABLE
- Object
- ObjectSpace
- Observable
- Observer
- PGError
- PGconn
- PGlarge
- PGresult
- PLATFORM

- FileTest
- FileUtils
- Fixnum
- Float
- FloatDomainError
- GC
- Gem
- GetoptLong
- Hash
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- Inflector
- Integer
- Interrupt
- Kernel
- LN_SUPPORTED
- LoadError
- LocalJumpError
- Logger
- Marshal
- MatchData
- MatchingData
- Math
- Method
- Module
- Mutex
- Mysql
- MysqlError
- MysqlField
- MysqlRes
- NIL
- NameError
- NilClass
- NoMemoryError
- NoMethodError
- NoWrite
- NotImplementedError
- Numeric
- OPT_TABLE
- Object
- ObjectSpace
- Observable
- Observer
- PGError
- PGconn
- PGlarge
- PGresult
- PLATFORM

- PStore
- ParseDate
- Precision
- Proc
- Process
- Queue
- RAKEVERSION
- RELEASE_DATE
- RUBY
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- Rack
- Rake
- RakeApp
- RakeFileUtils
- Range
- RangeError
- Rational
- Regexp
- RegexpError
- Request
- RuntimeError
- STDERR
- STDIN
- STDOUT
- ScanError
- ScriptError
- SecurityError
- Signal
- SignalException
- SimpleDelegater
- SimpleDelegator
- Singleton
- SizedQueue
- Socket
- SocketError
- StandardError
- String
- StringScanner
- Struct
- Symbol
- SyntaxError
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPSocket
- TCPserver
- TCPsocket
- TOPLEVEL_BINDING

- PStore
- ParseDate
- Precision
- Proc
- Process
- Queue
- RAKEVERSION
- RELEASE_DATE
- RUBY
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- Rack
- Rake
- RakeApp
- RakeFileUtils
- Range
- RangeError
- Rational
- Regexp
- RegexpError
- Request
- RuntimeError
- STDERR
- STDIN
- STDOUT
- ScanError
- ScriptError
- SecurityError
- Signal
- SignalException
- SimpleDelegater
- SimpleDelegator
- Singleton
- SizedQueue
- Socket
- SocketError
- StandardError
- String
- StringScanner
- Struct
- Symbol
- SyntaxError
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPSocket
- TCPserver
- TCPsocket
- TOPLEVEL_BINDING

- TRUE
- 任务
- 文本
- 线程
- 线程错误
- 线程组
- 时间
- 事务
- 真类
- 类型错误
- UDP套接字
- UDPsocket
- UNIX服务器
- UNIX套接字
- UNIXserver
- UNIXsocket
- 未绑定方法
- 网址
- 版本
- 详细模式
- YAML
- 除零错误
- @base_path
- 接受
- 访问
- 轴
- 行动
- 属性
- 应用2
- 回调
- 类别
- 连接
- 数据库
- 调度器
- 显示1
- 驱动
- 错误
- 格式
- 主机
- 键
- 布局
- 加载
- 链接
- 新建
- 通知
- 打开
- 公开
- 引用
- 渲染
- 请求
- 记录

- TRUE
- Task
- Text
- Thread
- ThreadError
- ThreadGroup
- Time
- Transaction
- TrueClass
- TypeError
- UDPSocket
- UDPsocket
- UNIXServer
- UNIXSocket
- UNIXserver
- UNIXsocket
- UnboundMethod
- Url
- VERSION
- Verbose
- YAML
- ZeroDivisionError
- @base_path
- accept
- Acces
- Axi
- action
- attributes
- application2
- callback
- category
- connection
- database
- dispatcher
- display1
- drive
- errors
- format
- host
- key
- layout
- load
- link
- new
- notify
- open
- public
- quote
- render
- request
- records

- 响应
- 保存
- 范围
- 发送
- 会话
- 系统
- 模板
- 测试
- 超时
- to_s
- 类型
- URI
- 访问次数
- 观察者

数据库字段名

- 创建时间
- 创建日期
- 更新时间
- 更新日期
- 删除时间
- （paranoia
- gem）
- 锁版本
- 类型
- 编号
- #{table_name}_count
- 位置
- 父编号
- 左值
- 右值
- 引用值

Ruby 保留字

- 别名
- 和
- BEGIN
- begin
- break
- case
- class
- def
- defined?
- do
- 否则
- 否则如果
- 结束
- 结束
- 确保
- 假

- responses
- save
- scope
- send
- session
- system
- template
- test
- timeout
- to_s
- type
- URI
- visits
- Observer

Database Field Names

- created_at
- created_on
- updated_at
- updated_on
- deleted_at
- (paranoia
- gem)
- lock_version
- type
- id
- #{table_name}_count
- position
- parent_id
- lft
- rgt
- quote_value

Ruby Reserved Words

- alias
- and
- BEGIN
- begin
- break
- case
- class
- def
- defined?
- do
- else
- elsif
- END
- end
- ensure
- false

- for
- if
- 模块
- 下一个
- nil
- not
- 或者
- 重做
- 救援
- 重试
- 返回
- 自身
- 父类然
- 后真未
- 定义除
- 非直到
- 当当...
- 时当...
- 期间生
- 成_ 文
- 件 __
- 行号 _
-
-
-

- for
- if
- module
- next
- nil
- not
- or
- redo
- rescue
- retry
- return
- self
- super
- then
- true
- undef
- unless
- until
- when
- while
- yield
- _ *FILE* _
- _ *LINE* _

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供此内容，
更多更改可发送至web@petercv.com以发布或更新新内容

阿卜杜拉	第7、12和13章
abhas	第40章
Abhinay	第10章
亚当·拉塞克	第2、5、6、9、12、13、38、40和42章
Ahsan Mahmood	第25章
Aigars Cibuļskis	第5章
阿贾伊·巴罗特	第1章和第13章
阿克谢·博拉德	第57章
阿里·马苏迪安普尔	第16章
安迪·盖奇	第2章
AnoE	第3章
安塔尔·伯德	第10章
阿尔斯兰·阿里	第54章
代码艺术	第20章
阿希什·比斯塔	第41章
阿斯瓦蒂	第1章
阿图尔·坎杜里	第7章和第14章
Avdept	第13章
阿瓦伊斯·沙夫卡特	第35章
B 刘	第37章
B8vrede	第33章
比贾尔·加贾尔	第3章
br3nt	第7章和第13章
布莱恩	第58章
布伦	第5章
cl3m	第2章
美食黑客	第20章
西里尔·杜雄	第2章和第17章
D	第3章
德内斯·帕普	第32章
danirod	第4章
达尔潘·查特拉瓦拉	第1章和第48章
达尔尚·帕特尔	第1章
DawnPaladin	第65章
Deep	第21章、第27章和第47章
迪帕克·卡布尔	第69章
迪帕克·马哈卡莱	第1、2、5和12章
dgilperez	第10、12和25章
达拉姆·戈拉普迪	第2、15和32章
dnsh	第13章
dodo121	第12章
唐·乔瓦尼	第43章
elasticman	第4章
埃姆雷·库尔特	第73章
esthervillars	第2章
法比奥·罗斯	第2、13和46章
fiedl	第18章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

Abdullah	Chapters 7, 12 and 13
abhas	Chapter 40
Abhinay	Chapter 10
Adam Lassek	Chapters 2, 5, 6, 9, 12, 13, 38, 40 and 42
Ahsan Mahmood	Chapter 25
Aigars Cibuļskis	Chapter 5
Ajay Barot	Chapters 1 and 13
Akshay Borade	Chapter 57
Ali MasudianPour	Chapter 16
Andy Gauge	Chapter 2
AnoE	Chapter 3
Antarr Byrd	Chapter 10
Arslan Ali	Chapter 54
ArtOfCode	Chapter 20
Ashish Bista	Chapter 41
Aswathy	Chapter 1
Atul Khanduri	Chapters 7 and 14
Avdept	Chapter 13
Awais Shafqat	Chapter 35
B Liu	Chapter 37
B8vrede	Chapter 33
Bijal Gajjar	Chapter 3
br3nt	Chapters 7 and 13
Brian	Chapter 58
buren	Chapter 5
cl3m	Chapter 2
Cuisine Hacker	Chapter 20
Cyril Duchon	Chapters 2 and 17
D	Chapter 3
Dénes Papp	Chapter 32
danirod	Chapter 4
Darpan Chhatravala	Chapters 1 and 48
Darshan Patel	Chapter 1
DawnPaladin	Chapter 65
Deep	Chapters 21, 27 and 47
Deepak Kabbur	Chapter 69
Deepak Mahakale	Chapters 1, 2, 5 and 12
dgilperez	Chapters 10, 12 and 25
Dharam Gollapudi	Chapters 2, 15 and 32
dnsh	Chapter 13
dodo121	Chapter 12
Don Giovanni	Chapter 43
elasticman	Chapter 4
Emre Kurt	Chapter 73
esthervillars	Chapter 2
Fabio Ros	Chapters 2, 13 and 46
fiedl	Chapter 18

火	第2章和第21章
弗兰比诺	第7章
傻瓜	第56章
弗朗西斯科·卢波·伦齐	第2、5、13、17和21章
fybw id	第29章
加斯顿	第6章
杰弗鲁瓦	第1章
格拉普沃斯	第3章
gwcodes	第17章
哈迪斯	第47章
哈迪克·坎贾里亚 ツ	第5章
哈迪克·乌帕迪亚	第6、11、12和15章
hgsongra	第7章和第34章
Hizqeel	第2章
HParker	第47章、第50章和第51章
hschin	第5章、第27章、第36章和第37章
HungryCoder	第2章
Ich	第8章
inye	第6章
jackerman09	第14、15和21章
jeffdill2	第5和13章
杰里米·格林	第15章
jkdev	第2章
乔尔·德拉珀	第3章
乔恩·伍德	第2章
豪尔赫·纳赫拉 T	第17章
Kelseydh	第32章
凯文·西尔维斯特（Kevin Sylvestre）	第2章
kfrz	第1章
阮庆（Khanh Pham）	第11章和第20章
基兰·安德鲁斯（Kieran Andrews）	第2章、第10章和第22章
金莫·欣蒂卡（Kimmo Hintikka）	第44章
柯蒂·托拉特（Kirti Thorat）	第1、2和5章
库尔金	第2、5和22章
拉希鲁	第17章
洛梅芬	第49章
洛伦佐·巴拉奇	第1章
吕克·布瓦塞	第43章
卢卡·克尔	第1、3、4、6、7、10和11章
马尔滕范弗利特	第3章和第5章
马尼什·阿加瓦尔	第5章和第11章
marcamillion	第3章和第15章
马里奥·乌赫尔	第2章
马克	第63章
马尔科·卡坎斯基	第7章
马尤尔·沙阿	第60章
ma_il	第6章和第27章
michaelpri	第1章和第30章
米哈伊	第28章
MikeAndr	第13章
米林德	第2章和第45章
米洛·P	第5章
米奇·范杜因	第67章

Fire	Chapters 2 and 21
Flambino	Chapter 7
fool	Chapter 56
Francesco Lupo Renzi	Chapters 2, 5, 13, 17 and 21
fybw id	Chapter 29
Gaston	Chapter 6
Geoffroy	Chapter 1
glapworth	Chapter 3
gwcodes	Chapter 17
hadees	Chapter 47
Hardik Kanjariya ツ	Chapter 5
Hardik Upadhyay	Chapters 6, 11, 12 and 15
hgsongra	Chapters 7 and 34
Hizqeel	Chapter 2
HParker	Chapters 47, 50 and 51
hschin	Chapters 5, 27, 36 and 37
HungryCoder	Chapter 2
Ich	Chapter 8
inye	Chapter 6
jackerman09	Chapters 14, 15 and 21
jeffdill2	Chapters 5 and 13
Jeremy Green	Chapter 15
jkdev	Chapter 2
Joel Drapper	Chapter 3
Jon Wood	Chapter 2
Jorge Najera T	Chapter 17
Kelseydh	Chapter 32
Kevin Sylvestre	Chapter 2
kfrz	Chapter 1
Khanh Pham	Chapters 11 and 20
Kieran Andrews	Chapters 2, 10 and 22
Kimmo Hintikka	Chapter 44
Kirti Thorat	Chapters 1, 2 and 5
KULKING	Chapters 2, 5 and 22
Lahiru	Chapter 17
Lomefin	Chapter 49
Lorenzo Baracchi	Chapter 1
Luc Boissaye	Chapter 43
Luka Kerr	Chapters 1, 3, 4, 6, 7, 10 and 11
maartenvanvliet	Chapters 3 and 5
Manish Agarwal	Chapters 5 and 11
marcamillion	Chapters 3 and 15
Mario Uher	Chapter 2
Mark	Chapter 63
Marko Kacanski	Chapter 7
Mayur Shah	Chapter 60
ma_il	Chapters 6 and 27
michaelpri	Chapters 1 and 30
Mihai	Chapter 28
MikeAndr	Chapter 13
Milind	Chapters 2 and 45
Milo P	Chapter 5
Mitch VanDuyn	Chapter 67

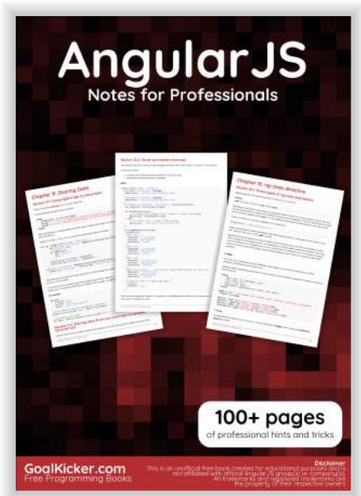
mlabarca	第25章
MMachinegun	第4章和第27章
mmichael	第12章
穆阿兹·拉菲	第6章和第31章
nifCody	第1章
尼扬塔	第11章
nomatteus	第5章和第15章
nuclearpidgeon	第2章
olive_tree	第1章
奥马尔·阿里	第58章和第59章
pablofullana	第12章
pastullo	第2章
powerup7	第3、6、13和15章
ppascualv	第71和72章
Pragash	第4章
RADan	第1章
拉斐尔·科斯塔	第55章
拉胡尔·辛格	第2章
雷诺·匡	第2和11章
rdnewman	第13章
重启	第5和13章
Reub	第15章
理查德·汉密尔顿	第1、2、5、12、14、15、22和39章
rii	第5章
罗宾	第2、5、13和29章
罗德里戈·阿尔古梅多	第5和14章
rogerdpack	第2章
rony36	第5章和第6章
罗里·奥凯恩	第2章和第5章
鲁斯兰	第61章
瑞安·希尔伯特	第2章
瑞安·K	第2章、第23章和第26章
sa77	第1章、第13章和第32章
saadlulu	第1章
萨普纳·金达尔	第11章和第62章
萨蒂什库马尔·贾亚拉杰	第1章和第70章
斯科特·马修曼	第41章
Sebastialonso	第3章
谢尔盖·赫梅列夫斯科伊	第19章
希瓦苏布拉马尼安·A	第53章
西尔维乌·西梅里亚	第2章
西蒙·曾	第39章
西蒙娜·卡莱蒂	第3章和第41章
斯莱迪	第8章
斯拉瓦·K	第61章
索海尔·哈利勒	第2章
斯文·罗伊特	第3章、第7章和第21章
tes	第26章
tessi	第5章
阮良胜	第64章
理论的	第1章
蒂亚戈·阿劳若	第24章
tirdadc	第52章

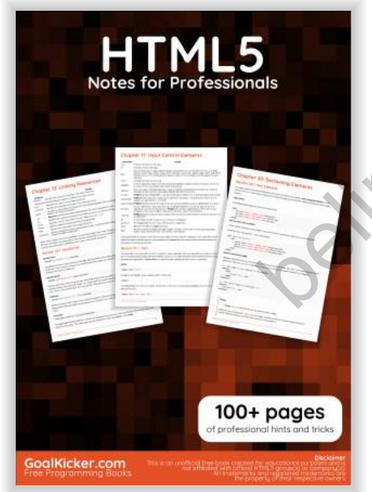
mlabarca	Chapter 25
MMachinegun	Chapters 4 and 27
mmichael	Chapter 12
Muaaz Rafi	Chapters 6 and 31
nifCody	Chapter 1
Niyanta	Chapter 11
nomatteus	Chapters 5 and 15
nuclearpidgeon	Chapter 2
olive_tree	Chapter 1
Omar Ali	Chapters 58 and 59
pablofullana	Chapter 12
pastullo	Chapter 2
powerup7	Chapters 3, 6, 13 and 15
ppascualv	Chapters 71 and 72
Pragash	Chapter 4
RADan	Chapter 1
Rafael Costa	Chapter 55
Rahul Singh	Chapter 2
Raynor Kuang	Chapters 2 and 11
rdnewman	Chapter 13
Reboot	Chapters 5 and 13
Reub	Chapter 15
Richard Hamilton	Chapters 1, 2, 5, 12, 14, 15, 22 and 39
rii	Chapter 5
Robin	Chapters 2, 5, 13 and 29
Rodrigo Argumedo	Chapters 5 and 14
rogerdpack	Chapter 2
rony36	Chapters 5 and 6
Rory O'Kane	Chapters 2 and 5
Ruslan	Chapter 61
Ryan Hilbert	Chapter 2
Ryan K	Chapters 2, 23 and 26
sa77	Chapters 1, 13 and 32
saadlulu	Chapter 1
Sapna Jindal	Chapters 11 and 62
Sathishkumar Jayaraj	Chapters 1 and 70
Scott Matthewman	Chapter 41
Sebastialonso	Chapter 3
Sergey Khmelevskoy	Chapter 19
Shivasubramanian A	Chapter 53
Silviu Simeria	Chapter 2
Simon Tsang	Chapter 39
Simone Carletti	Chapters 3 and 41
Sladey	Chapter 8
Slava.K	Chapter 61
sohail khalil	Chapter 2
Sven Reuter	Chapters 3, 7 and 21
tes	Chapter 26
tessi	Chapter 5
Thang Le Sy	Chapter 64
theoretisch	Chapter 1
thiago araujo	Chapter 24
tirdadc	Chapter 52

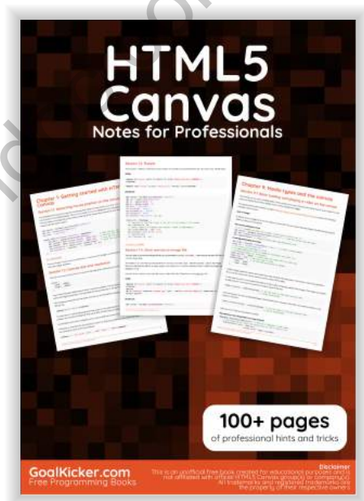
泰坦	第32章
图布尔克	第61章
乌玛尔·汗	第33章
撤销	第2章和第8章
乌扎伊夫	第1章和第5章
维沙尔·塔吉 PM	第13章和第66章
弗拉基米尔·巴利茨基	第1章、第4章和第11章
Whitecat	第68章
威廉·罗梅罗	第14章
李新阳	第7章
雅娜	第1章
佐兰	第2章
孙悟空	第2章

titan	Chapter 32
toobulkeh	Chapter 61
Umar Khan	Chapter 33
Undo	Chapters 2 and 8
uzaif	Chapters 1 and 5
Vishal Taj PM	Chapters 13 and 66
Volodymyr Balytskyy	Chapters 1, 4 and 11
Whitecat	Chapter 68
William Romero	Chapter 14
Xinyang Li	Chapter 7
Yana	Chapter 1
Zoran	Chapter 2
孙悟空	Chapter 2

你可能还喜欢







You may also like