



Бесплатная электронная книга

УЧУСЬ

# Ruby on Rails

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#ruby-on-  
rails

.....	1
<b>1: Ruby on Rails</b> .....	<b>2</b>
.....	2
.....	2
Examples.....	3
Ruby on Rails.....	3
Rails     RSpec.....	5
.....	6
.....	7
Rails .....	8
Rails- JSON.....	8
Rails.....	9
<b>2: ActionCable</b> .....	<b>12</b>
.....	12
Examples.....	12
[] .....	12
[] (CoffeeScript).....	12
<b>// JavaScripts // notifications.coffee</b> .....	<b>12</b>
<b>app / assets / javascripts / application.js #</b> .....	<b>12</b>
<b>app / assets / javascripts / cable.js #</b> .....	<b>13</b>
.....	13
<b>3: ActionController</b> .....	<b>14</b>
.....	14
Examples.....	14
JSON HTML.....	14
().....	15
.....	15
(Basic).....	15
.....	16
.....	17
404, .....	18

REST .....	18
.....	19
.....	20
.....	23
Rescuing ActiveRecord :: RecordNotFound redirect_to.....	24
<b>4: ActionMailer .....</b>	<b>25</b>
.....	25
.....	25
Examples .....	25
Mailer.....	25
<b>user_mailer.rb .....</b>	<b>25</b>
<b>user.rb .....</b>	<b>26</b>
<b>approved.html.erb .....</b>	<b>26</b>
<b>approved.text.erb .....</b>	<b>26</b>
.....	26
.....	27
ActionMailer.....	27
.....	27
ActionMailer.....	34
<b>5: ActiveJob .....</b>	<b>36</b>
.....	36
Examples.....	36
.....	36
.....	36
<b>6: ActiveRecord .....</b>	<b>37</b>
.....	37
Examples.....	37
ActiveModel :: Validations.....	37
<b>7: ActiveRecord .....</b>	<b>38</b>
Examples.....	38
.....	38
.....	

..... 39

/ ..... 39

..... 40

..... 40

..... 40

..... 41

..... 42

..... 43

..... 43

**8: ActiveSupport..... 44**

..... 44

Examples..... 44

  : ..... 44

**# ..... 44**

**# ..... 44**

**# ..... 44**

**# ..... 45**

**# ..... 45**

  : / ..... 45

**# TO\_TIME..... 45**

**# to\_date..... 45**

**# to\_datetime..... 46**

  : ..... 46

**# ?..... 46**

  : ..... 46

**# ..... 46**

**# ..... 46**

**# ..... 47**

**# truncate\_words..... 47**

<b># strip_heredoc</b> .....	<b>47</b>
: String Inflection.....	48
<b>#</b> .....	<b>48</b>
<b>#</b> .....	<b>48</b>
<b># constantize</b> .....	<b>49</b>
<b># safe_constantize</b> .....	<b>49</b>
<b># camelize</b> .....	<b>49</b>
<b># titleize</b> .....	<b>49</b>
<b>#</b> .....	<b>50</b>
<b># dasherize</b> .....	<b>50</b>
<b># demodulize</b> .....	<b>50</b>
<b># deconstantize</b> .....	<b>50</b>
<b>#</b> .....	<b>50</b>
<b># tableize</b> .....	<b>51</b>
<b>#</b> .....	<b>51</b>
<b># Humanize</b> .....	<b>51</b>
<b># upcase_first</b> .....	<b>52</b>
<b># foreign_key</b> .....	<b>52</b>
<b>9: API Rails</b> .....	<b>53</b>
Examples.....	53
API-.....	53
<b>10: Elasticsearch</b> .....	<b>54</b>
Examples.....	54
.....	54
.....	54
.....	55
Searchkick.....	55
<b>11: I18n -</b> .....	<b>57</b>
.....	57
Examples.....	57

l18n .....	57
l18n .....	57
.....	58
.....	58
<b>URL .....</b>	<b>59</b>
.....	59
<b>Default Locale .....</b>	<b>60</b>
HTTP-.....	61
.....	61
1. ....	62
2. CloudFlare.....	62
ActiveRecord.....	62
l18n HTML- .....	64
<b>12: Mongoid.....</b>	<b>65</b>
Examples.....	65
.....	65
.....	65
.....	66
.....	66
.....	66
.....	67
.....	67
<b>13: Rails Cookbook - / .....</b>	<b>69</b>
Examples.....	69
rails.....	69
Rails - .....	70
- `'#.....	70
<b>14: Rails Engine - .....</b>	<b>72</b>
.....	72
.....	72
Examples.....	72
.....	72
.....	73

<b>15: Rails -Engines</b>	<b>76</b>
.....	76
.....	76
.....	76
.....	76
Examples	76
.....	76
<b>16: Rails logger</b>	<b>78</b>
Examples	78
Rails.logger	78
<b>17: RSpec Ruby on Rails</b>	<b>80</b>
.....	80
Examples	80
RSpec	80
<b>18: Turbolinks</b>	<b>82</b>
.....	82
.....	82
:	<b>82</b>
Examples	82
turbolink	82
.....	83
:	<b>83</b>
.....	83
.....	84
:	<b>84</b>
.....	85
<b>19: Rails 5 API</b>	<b>86</b>
Examples	86
Rails authenticate_with_http_token	86
<b>20: CanCan</b>	<b>87</b>
.....	87

.....	87
Examples.....	87
CanCan.....	87
.....	88
.....	88
.....	90
<b>21:</b> .....	<b>91</b>
Examples.....	91
.....	91
.....	91
.....	91
<b>22: ActiveRecord</b> .....	<b>92</b>
.....	92
Examples.....	92
.....	92
ActiveRecord .....	92
.....	93
.....	93
Callbacks.....	93
.....	94
<b>23: ActiveRecord</b> .....	<b>95</b>
.....	95
Examples.....	95
.....	95
<b>24: ActiveRecord</b> .....	<b>97</b>
Examples.....	97
.....	97
has_one.....	97
.....	98
.....	98
Has_many: .....	99
Has_one: .....	99



has_and_belongs_to_many.....	100
.....	100
<b>25: Api Devise.....</b>	<b>102</b>
.....	102
Examples.....	102
.....	102
.....	103
<b>26: Rails.....</b>	<b>105</b>
.....	105
.....	105
Examples.....	105
.....	105
.....	106
.....	106
OmniAuth.....	107
has_secure_password.....	107
.....	107
has_secure_password .....	107
has_secure_token.....	107
<b>27: .....</b>	<b>109</b>
Examples.....	109
safe_constantize.....	109
safe_constantize.....	109
<b>28: .....</b>	<b>110</b>
.....	110
Examples.....	110
Figaro.....	110
<b>29: ActiveRecord.....</b>	<b>112</b>
Examples.....	112
.....	112
.....	112
<b>30: Ruby on Rails.....</b>	<b>113</b>

Examples.....	113
Ruby on Rails.....	113
<b>31:</b> .....	<b>115</b>
.....	115
Examples.....	115
SimpleDelegator.....	115
Draper.....	116
<b>32:</b> .....	<b>117</b>
.....	117
.....	117
.....	117
Examples.....	117
, rails_admin ge.....	117
<b>33: RDS Amazon</b> .....	<b>121</b>
.....	121
Examples.....	121
, MYSQL RDS .....	121
<b>34:</b> .....	<b>123</b>
.....	123
Gemfile.....	123
Examples.....	123
?.....	123
<b>Rails</b> .....	<b>123</b>
Gemfile.....	123
Gemfile.lock.....	123
.....	<b>124</b>
Bundler.....	124
Gemfiles.....	125
Gemsets.....	126
<b>35:</b> .....	<b>129</b>
Examples.....	129

Carrierwave.....	129
- .....	129
<b>36:</b> .....	<b>132</b>
.....	132
Examples.....	132
.....	132
<b>37:</b> .....	<b>139</b>
.....	139
Examples.....	139
Rails Quickstart.....	139
<b>Gemfile</b> .....	<b>139</b>
<b>/ models / user.rb</b> .....	<b>139</b>
<b>h11</b> .....	<b>139</b>
<b>h12</b> .....	<b>139</b>
<b>38: Rails</b> .....	<b>141</b>
.....	141
Examples.....	141
.....	141
.....	141
<b>39:</b> .....	<b>142</b>
.....	142
Examples.....	142
Rails, Active Record UTC.....	143
Rails .....	143
<b>40: CSV-</b> .....	<b>144</b>
.....	144
Examples.....	144
CSV .....	144
<b>41: Ruby on Rails</b> .....	<b>146</b>
.....	146
Examples.....	146

, ; .....	146
<b>42: React.js Hyperloop .....</b>	<b>148</b>
.....	148
.....	148
Examples .....	148
( ) Rails .....	148
() .....	149
HTML- .....	149
.....	149
.....	150
Callbacks .....	150
<b>43: ActiveRecord .....</b>	<b>151</b>
.....	151
Examples .....	151
.....	151
.....	152
.....	152
where.not .....	153
.....	153
ActiveRecord Bang (!) .....	154
.find_by .....	155
.....	155
ActiveRecord .....	155
.....	156
.count .....	157
.distinct ( .uniq) .....	157
.....	158
.....	158
.....	159
<b>44: GoogleMaps Rails .....</b>	<b>160</b>
Examples .....	160
javascript Google Maps .....	160
.....	

161	
google	161
javascript	163
,	164
	165
	165
json	166
	<b>166</b>
	<b>167</b>
	<b>167</b>
<b>45: Rails</b>	<b>169</b>
	169
	169
	169
Examples	170
Rails Generate Model	170
Rails Generate Migration	170
Rails Generate Scaffold	171
Rails	172
<b>46:</b>	<b>174</b>
	174
Examples	174
-	174
	174
	175
<b>47:</b>	<b>176</b>
Examples	176
	176
<b>48:</b>	<b>178</b>
Examples	178
Rails	178
	178

Rails.....	179
.....	179
.....	180
<b>49: PDF.....</b>	<b>182</b>
Examples.....	182
.....	182
.....	183
.....	<b>183</b>
.....	<b>183</b>
.....	<b>183</b>
<b>50: .....</b>	<b>184</b>
Examples.....	184
.....	184
SQL.....	184
.....	185
.....	186
HTTP-.....	186
.....	187
<b>51: .....</b>	<b>188</b>
.....	188
.....	188
Examples.....	188
().....	188
.....	190
.....	192
.....	195
.....	196
.....	197
URL- .....	197
.....	198
RESTful.....	198
.....	199
.....	

.....	199
.....	200
.....	200
<b>52: ActiveRecord.....</b>	<b>202</b>
.....	202
.....	202
Examples.....	203
.....	203
.....	203
.....	204
.....	204
.....	204
.....	204
.....	204
.....	205
.....	206
.....	206
.....	207
.....	207
.....	207
.....	208
.....	208
.....	208
.....	209
.....	209
.....	209
.....	210
.....	210
.....	211
hstore.....	211
.....	211
.....	212
NOT NULL .....	212

<b>53: ActiveRecord</b>	<b>214</b>
.....	214
Examples	214
.....	214
.....	214
.....	<b>214</b>
.....	<b>214</b>
<b>54: : AASM</b>	<b>216</b>
Examples	216
AASM	216
<b>55:</b>	<b>219</b>
.....	219
Examples	219
.....	219
.....	220
Rails	220
<b>56:</b>	<b>221</b>
Examples	221
101	221
<b>1. Rails</b>	<b>221</b>
<b>2: Turbolinks</b>	<b>221</b>
<b>3: AngularJS</b>	<b>221</b>
<b>4:</b>	<b>222</b>
<b>5:</b>	<b>222</b>
<b>57:</b>	<b>225</b>
Examples	225
1.	225
<b>58:</b>	<b>226</b>
Examples	226
Rails 4.2 Rails 5.0	226
<b>59:</b>	<b>228</b>



.....	228
Examples.....	228
.....	228
.....	229
<b>60:</b> .....	<b>232</b>
Examples.....	232
Rails.....	232
IDE.....	232
Ruby on Rails + .....	234
<b>Ruby / Rails :</b> .....	<b>234</b>
1. : Exception .inspect .....	234
2. : IRB byebug pry.....	234
.....	<b>235</b>
, Ruby, :.....	235
ruby-on-rails pry.....	236
<b>61: ActiveRecord</b> .....	<b>239</b>
Examples.....	239
.....	239
.....	240
.....	240
.....	241
.....	241
.....	241
.....	241
.....	242
ActiveModel::Validator validates_with.....	242
ActiveModel::EachValidator validate.....	242
.....	243
.....	243
.....	243
.....	244
.....	245
<b>62:</b> .....	<b>246</b>

Examples.....	246
Partials.....	246
.....	<b>246</b>
.....	<b>247</b>
AssetTagHelper.....	247
.....	<b>247</b>
image_path.....	247
URL.....	247
IMAGE_TAG.....	247
<b>JavaScript.....</b>	<b>247</b>
javascript_include_tag.....	248
javascript_path.....	248
javascript_url.....	248
.....	<b>248</b>
stylesheet_link_tag.....	248
stylesheet_path.....	248
stylesheet_url.....	248
.....	<b>248</b>
.....	249
HTML- Views.....	250
HAML - .....	250
<b>63: Rails Heroku.....</b>	<b>253</b>
Examples.....	253
.....	253
Heroku.....	256
<b>64: Rails .....</b>	<b>258</b>
.....	258
Examples.....	258
, Rails?.....	258
Rails Rails 1.x.....	258
Rails Rails 2.x.....	258

Rails Rails 3.x.....	258
<b>65: , .....</b>	<b>260</b>
Examples.....	260
Rails rails_react gem.....	260
response_rails .....	260
.....	261
<b>66: Rails.....</b>	<b>263</b>
Examples.....	263
(DRY).....	263
.....	263
, .....	264
default_scope.....	265
default_scope order.....	265
<b>default_scope .....</b>	<b>266</b>
<b>unscoped.....</b>	<b>266</b>
<b>unscoped .....</b>	<b>266</b>
<b>default_scope.....</b>	<b>267</b>
(YAGNI).....	267
.....	268
.....	268
Bloat.....	268
.....	268
.....	268
.....	268
.....	268
KISS - , .....	268
YAGNI - .....	269
.....	269
( ).....	269
<b>67: 5.....</b>	<b>272</b>
Examples.....	272
API Ruby on Rails 5.....	272

Ruby on Rails 5 RVM.....	274
<b>68:</b> .....	<b>275</b>
.....	275
Examples.....	275
-.....	275
<b>69:</b> .....	<b>277</b>
.....	277
Examples.....	277
.....	277
<b>70:</b> .....	<b>278</b>
Examples.....	278
.....	278
.....	278
.....	278
.....	279
.....	280
<b>71: Rails-</b> .....	<b>281</b>
Examples.....	281
.....	281
.....	281
<b>72:</b> .....	<b>282</b>
Examples.....	282
.....	282
<b>73:</b> .....	<b>283</b>
.....	283
.....	283
Examples.....	283
.....	283
.....	283
.....	284
.....	284
-.....	284

.....	284
.....	285
.....	285
.....	285
.....	285
.....	285
.....	285
.....	286
<b>74:</b> .....	<b>287</b>
.....	287
.....	287
Examples .....	287
.....	287
<b>Stripe</b> .....	<b>287</b>
<b>Stripe</b> .....	<b>288</b>
.....	288
.....	288
.....	290

---

# Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [ruby-on-rails](#)

It is an unofficial and free Ruby on Rails ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Ruby on Rails.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# глава 1: Начало работы с Ruby on Rails

## замечания



Ruby on Rails (RoR) или Rails - это популярная платформа веб-приложений с открытым исходным кодом. Rails использует Ruby, HTML, CSS и JavaScript для создания веб-приложения, работающего на веб-сервере. Rails использует шаблон model-view-controller (MVC) и обеспечивает полный набор библиотек из базы данных вплоть до представления.

## Версии

Версия	Дата выхода
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

Версия	Дата выхода
1,1	2006-03-28
1,0	2005-12-13

## Examples

### Создание приложения Ruby on Rails

В этом примере предполагается, что *Ruby* и *Ruby on Rails* уже установлены правильно. Если нет, вы можете найти, как это сделать [здесь](#).

Откройте командную строку или терминал. Чтобы создать новое приложение rails, используйте **новую** команду **rails**, а затем имя вашего приложения:

```
$ rails new my_app
```

Если вы хотите создать приложение Rails с определенной версией Rails, вы можете указать его во время создания приложения. Для этого используйте `rails _version_ new` а затем имя приложения:

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

Это создаст приложение Rails под названием `MyApp` в каталоге `my_app` и установит зависимости `gem`, которые уже упоминаются в `Gemfile` используя `bundle install`.

Чтобы переключиться на каталог вновь созданного приложения, используйте команду `cd`, которая обозначает `change directory`.

```
$ cd my_app
```

В `my_app` есть несколько автоматически сгенерированных файлов и папок, которые составляют структуру приложения Rails. Ниже приведен список файлов и папок, созданных по умолчанию:

Папка	Цель
приложение/	Содержит контроллеры, модели, представления, помощники, почтовые программы и активы для вашего приложения.
бен /	Содержит скрипт rails, который запускает ваше приложение и может содержать другие скрипты, которые вы используете для настройки, обновления, развертывания или запуска приложения.



Папка	Цель
конфиг /	Настройте маршруты, базу данных и многое другое.
config.ru	Конфигурация стойки для серверов на стойке, используемых для запуска приложения.
дБ /	Содержит вашу текущую схему базы данных, а также миграцию базы данных.
Gemfile Gemfile.lock	Эти файлы позволяют указать, какие зависимости от gem необходимы для вашего приложения Rails. Эти файлы используются жемчужиной Bundler.
Библиотека /	Расширенные модули для вашего приложения.
журнал/	Файлы журналов приложений.
общественности /	Единственная папка, которую видит мир как есть. Содержит статические файлы и скомпилированные активы.
Rakefile	Этот файл находит и загружает задачи, которые могут выполняться из командной строки. Определения задач определяются во всех компонентах Rails.
README.md	Это краткое руководство для вашего приложения. Вы должны отредактировать этот файл, чтобы сообщить другим, что делает ваше приложение, как его настроить и т. Д.
тестовое задание/	Единичные испытания, приборы и другие испытательные устройства.
темп /	Временные файлы (например, файлы кеша и pid).
продавец /	Место для всех сторонних кодов. В типичном приложении Rails это включает в себя драгоценные камни.

Теперь вам нужно создать базу данных из файла `database.yml` :

## 5.0

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

## 5.0

```
rake db:create
```

Теперь, когда мы создали базу данных, нам нужно запустить миграцию для настройки таблиц:

5.0

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

5.0

```
rake db:migrate
```

Чтобы запустить приложение, нам нужно запустить сервер:

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

По умолчанию рельсы запустит приложение на порту 3000. Чтобы запустить приложение с другим номером порта, нам нужно запустить сервер, например,

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

Если вы перейдете в <http://localhost:3000> в своем браузере, вы увидите страницу приветствия Rails, показывающую, что ваше приложение запущено.

Если он вызывает ошибку, может быть несколько возможных проблем:

- Существует проблема с `config/database.yml`
- У вас есть зависимости в вашем `Gemfile`, которые не были установлены.
- У вас есть ожидающие миграции. Run `rails db:migrate`
- Если вы перейдете к предыдущим маршрутам переноса `rails db:rollback`

Если это все еще вызывает ошибку, вы должны проверить свой `config/database.yml`

## Создайте новое приложение Rails с выбором базы данных и включая инструмент проверки RSpec

Rails использует `sqlite3` как базу данных по умолчанию, но вы можете создать новое приложение rails с базой данных по вашему выбору. Просто добавьте параметр `-d` а затем имя базы данных.

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

Это (неисчерпывающий) список доступных параметров базы данных:

- MySQL
- оракул
- PostgreSQL
- sqlite3
- FrontBase
- IBM\_DB
- SQLServer
- jdbcmysql
- jdbcsqlite3
- jdbcpostgresql
- JDBC

Команда `-T` указывает, чтобы пропустить установку `minitest`. Чтобы установить альтернативный набор тестов, например [RSpec](#), отредактируйте `Gemfile` и добавьте

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

Затем запустите с консоли следующую команду:

```
rails generate rspec:install
```

## Генерирование контроллера

Чтобы создать контроллер (например, «`Posts`»), перейдите в каталог проекта из командной строки или терминала и запустите:

```
$ rails generate controller Posts
```

Вы можете сократить этот код, заменив `generate` на `g`, например:

```
$ rails g controller Posts
```

Если вы откроете вновь созданное приложение / контроллеры / **`posts_controller.rb`**, вы увидите контроллер без каких-либо действий:

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

Можно создать методы по умолчанию для контроллера, передав аргументы имени контроллера.

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

Чтобы создать контроллер внутри модуля, укажите имя контроллера как путь, например `parent_module/controller_name` . Например:

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

Это приведет к созданию следующих файлов:

```
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
```

Контроллер - это просто класс, который определяется как наследуемый от `ApplicationController` .

Внутри этого класса вы определяете методы, которые станут действиями для этого контроллера.

## Создание ресурса с помощью лесов

От [guide.rubyonrails.org](http://guide.rubyonrails.org):

Вместо создания модели напрямую. , , давайте создадим эшафот. Эшафот в Rails - это полный набор моделей, миграция базы данных для этой модели, контроллер для управления им, представления для просмотра и управления данными и набор тестов для каждого из вышеперечисленных.

Ниже приведен пример создания леса под названием `Task` с именем строки и текстовым описанием:

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

Это приведет к созданию следующих файлов:

```
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
```

пример для удаления файлов, созданных эшафотом для ресурса под названием `Task`

```
rails destroy scaffold Task
```

## Создайте новое приложение Rails с нестандартным адаптером базы данных

Rails поставляется по умолчанию с помощью `ActiveRecord`, ORM (реляционное сопоставление объектов), полученного из шаблона с тем же именем.

Как ORM, он создан для обработки реляционного сопоставления, а точнее, для обработки запросов SQL для вас, отсюда ограничение только для баз данных SQL.

Однако вы все равно можете создать приложение Rails с другой системой управления базами данных:

1. просто создайте приложение без активной записи

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

2. добавить собственную систему управления базами данных в `Gemfile`

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

3. `bundle install` и выполнить шаги установки из нужной базы данных.

В этом примере `mongoid` - это сопоставление объектов для `MongoDB` и - как и многие другие камни базы данных, построенные для рельсов - он также наследует от `ActiveModel` же, как и `ActiveRecord`, который обеспечивает общий интерфейс для многих функций, таких как проверки, обратные вызовы, переводы и т. Д. ,

Другие адаптеры баз данных включают, но не ограничиваются:

- `DataMapper`
- сиквел рельсы

## Создание Rails-интерфейсов в JSON

В этом примере предполагается, что у вас есть опыт создания приложений Rails.

Чтобы создать приложение только для API в Rails 5, запустите

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

Добавить `active_model_serializers` в `Gemfile`

```
gem 'active_model_serializers'
```

## установить комплект в терминале

```
bundle install
```

Установите `ActiveModelSerializer` адаптер для использования `: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 )
```

## Создайте новый эшафот для вашего ресурса

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

Это приведет к созданию следующих файлов:

Контроллер: приложение / контроллеры / `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
```

## Установка Rails

### Установка Rails на Ubuntu

На чистом ubuntu установка Rails должна быть прямой

### Обновление пакетов Ubuntu

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

### Установка зависимостей Ruby и Rails

```
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
```

Установка рубинового менеджера версий. В этом случае простой использует `rbenv`

```
git clone https://github.com/rbenv/rbenv.git ~/.rbenv
```

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

## Установка 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
```

## Перезапустить оболочку

```
exec $SHELL
```

## Установить рубин

```
rbenv install 2.3.1
rbenv global 2.3.1
rbenv rehash
```

## Установка рельсов

```
gem install rails
```

## Установка Rails в Windows

### Шаг 1: *Установка Ruby*

Нам нужен язык программирования Ruby. Мы можем использовать предварительно скомпилированную версию Ruby с именем RubyInstaller.

- Загрузите и запустите Ruby Installer с [rubyinstaller.org](http://rubyinstaller.org).
- Запустите программу установки. Проверьте «Добавить исполняемые файлы Ruby на свой PATH», затем установите.
- Чтобы получить доступ к Ruby, перейдите в меню Windows, выберите «Все программы», прокрутите вниз до Ruby и нажмите «Start Command Prompt with Ruby». Откроется терминал командной строки. Если вы `ruby -v` и нажмите Enter, вы увидите номер версии Ruby, который вы установили.

### Шаг 2: *Комплект разработки Ruby*

После установки Ruby мы можем попытаться установить Rails. Но некоторые из библиотек Rails зависят от необходимости некоторых инструментов сборки для компиляции, а Windows не имеет этих инструментов по умолчанию. Вы можете определить это, если вы видите ошибку при попытке установить Rails `Gem::InstallError: The '[gem name]' native gem requires installed build tools.` Чтобы исправить это, нам нужно установить Ruby Development Kit.

- Загрузить [DevKit](#)

- Запустите программу установки.
- Нам нужно указать папку, в которой мы собираемся навсегда установить DevKit. Я рекомендую установить его в корень вашего жесткого диска в `C:\RubyDevKit` . (Не используйте пробелы в имени каталога.)

Теперь нам нужно сделать инструменты DevKit доступными для Ruby.

- В командной строке перейдите в каталог DevKit. `cd C:\RubyDevKit` или любой другой каталог, в который вы его установили.
- Нам нужно запустить скрипт Ruby для инициализации установки DevKit. Тип `ruby dk.rb init` . Теперь мы расскажем об этом же скрипте, чтобы добавить DevKit в нашу установку Ruby. Тип `ruby dk.rb install` .

Теперь DevKit будет доступен для ваших инструментов Ruby, которые будут использоваться при установке новых библиотек.

### Шаг 3: *Рельсы*

Теперь мы можем установить Rails. Rails - это драгоценный камень Ruby. В командной строке введите:

```
gem install rails
```

Как только вы нажмете Enter, программа `gem` будет загружать и устанавливать эту версию Rails-жемчужины вместе со всеми остальными драгоценными камнями.

### Шаг 4: *Node.js*

Некоторые библиотеки, от которых зависит Rails, требуют установки времени выполнения JavaScript. Давайте установим Node.js, чтобы эти библиотеки работали правильно.

- ВЫГРУЖАТЬ Node.js установки из [здесь](#) .
- Когда загрузка завершена, зайдите в папку для загрузки и запустите установщик `node-v4.4.7.pkg` .
- Прочтите полное лицензионное соглашение, примите условия и нажмите «Далее» через остальных мастеров, оставив все по умолчанию.
- Появится окно с вопросом, хотите ли вы разрешить приложению вносить изменения в свой компьютер. Нажмите «Да».
- Когда установка будет завершена, вам необходимо перезагрузить компьютер, чтобы Rails мог получить доступ к Node.js.

После перезагрузки компьютера не забудьте перейти в меню Windows, нажмите «Все программы», прокрутите вниз до Ruby и нажмите «Начать командную строку с Ruby».

Прочитайте Начало работы с Ruby on Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/225/начало-работы-с-ruby-on-rails>



---

# глава 2: ActionCable

## замечания

ActionCable был доступен для Rails 4.x и был включен в Rails 5. Он позволяет легко использовать websockets для обмена в реальном времени между сервером и клиентом.

## Examples

### [Основная] сторона сервера

```
# 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
```

### [Основная] Сторона клиента (Coffeescript)

---

## Приложение / активы / JavaScripts / каналы / 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)

  notify: (data)->
    @perform('notify', data: data)
```

## app / assets / javascripts / application.js # обычно генерируется как это

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

## app / assets / javascripts / cable.js # обычно генерируется как это

```
//= require action_cable
//= require_self
//= require_tree ./channels

(function() {
  this.App || (this.App = {});

  App.cable = ActionCable.createConsumer();

}).call(this);
```

## Аутентификация пользователя

```
# 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
```

Прочитайте ActionCable онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/1498/actioncable>

# глава 3: ActionController

## Вступление

Action Controller - это C в MVC. После того, как маршрутизатор определил, какой контроллер использовать для запроса, контроллер отвечает за определение запроса и вывод результата.

Контроллер получит запрос, выборку или сохранение данных из модели и использование представления для создания вывода. Контроллер можно рассматривать как посредника между моделями и представлениями. Это делает данные модели доступными для представления, чтобы он мог отображаться пользователю, и он сохраняет или обновляет данные пользователя в модели.

## Examples

### Вывод JSON вместо 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
```

Кроме того, вам понадобится маршрут:

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

Это будет реагировать двумя разными способами на запросы `/users` :

- Если вы посетили `/users` или `/users.html` , он покажет html-страницу с контентом `Hello World`
- Если вы посетите `/users.json` , он отобразит объект JSON, содержащий:

```
[
  {
    "name": "foo",
    "email": "foo@example.org"
  }
]
```

Вы можете **опустить** `format.html { render inline: "Hello World" }` если хотите, чтобы ваш

маршрут отвечал только на запросы JSON.

## Контроллеры (базовые)

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render html: "Hello World" }
    end
  end
end
```

Это базовый контроллер с добавлением следующего маршрута (в routes.rb):

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

Будет отображаться сообщение `Hello World` на веб-странице при доступе к URL `/users`

## параметры

Контроллеры имеют доступ к параметрам HTTP (вы можете знать их как `?name=foo` в URL-адресах, но Ruby on Rails обрабатывают и другие форматы!) И выводят на них разные ответы. Невозможно различать параметры GET и POST, но вы не должны этого делать в любом случае.

```
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
```

Как обычно, наш маршрут:

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

Получите доступ к URL `/users?name=john` и на выходе будет `Hello John`, `access`

`/users?name=whatever` и вывод будет `Hello someone`

## Параметры фильтрации (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

```

Вы можете разрешить (или отклонить) некоторые параметры, чтобы *пройти* только то, что вы хотите, и у вас не будет таких неприятных сюрпризов, как параметры пользовательских настроек, которые не должны быть изменены.

Visiting `/users?name=john&sentence=developer` покажет `Hello john developer`, однако посещение `/users?name=smith&sentence=spy` отобразит только `Hello smith`, потому что `:sentence` разрешено только при доступе к `john`

## Перенаправление

Предполагая маршрут:

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

Вы можете перенаправить на другой URL-адрес, используя:

```

class UsersController
  def index
    redirect_to "http://stackoverflow.com/"
  end
end

```

Вы можете вернуться к предыдущей странице, которую посетил пользователь, используя:

```
redirect_to :back
```

Обратите внимание, что в *Rails 5* синтаксис для перенаправления назад отличается:

```
redirect_back fallback_location: "http://stackoverflow.com/"
```

Что будет пытаться перенаправить на предыдущую страницу, и в случае, если это невозможно (браузер блокирует заголовок `HTTP_REFERER`), он перенаправляется на

```
:fallback_location
```

## Использование представлений

Предполагая маршрут:

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

И контроллер:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

Будет `app/users/index.html.erb` представление `app/users/index.html.erb` . Если вид:

```
Hello <strong>World</strong>
```

Результатом будет веб-страница с текстом: «Hello **World** »

Если вы хотите отобразить другое представление, вы можете использовать:

```
render "pages/home"
```

`app/views/pages/home.html.erb` этого будет использовано `app/views/pages/home.html.erb` .

Вы можете передавать переменные в представления с использованием переменных экземпляра контроллера:

```
class UsersController < ApplicationController
  def index
    @name = "john"

    respond_to do |format|
      format.html { render }
    end
  end
end
```

А в файле `app/views/users/index.html.erb` вы можете использовать `@name` :

```
Hello <strong><%= @name %></strong>
```

И выход будет: «Hello **john** »

Важное примечание вокруг синтаксиса рендеринга, вы можете полностью опустить

синтаксис `render` , Rails предполагает, что если вы его опустите. Так:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html { render }
    end
  end
end
```

Вместо этого можно написать:

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html
    end
  end
end
```

Rails достаточно умен, чтобы понять, что он должен отображать файл

`app/views/users/index.html.erb` .

## 404, когда запись не найдена

Rescue from record not found error вместо того, чтобы показывать исключение или белую страницу:

```
class ApplicationController < ActionController::Base

  # ... your other stuff here

  rescue_from ActiveRecord::RecordNotFound do |exception|
    redirect_to root_path, 404, alert: 'Record not found'
  end
end
```

## Базовый контроллер REST

```
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
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: '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

```

## Отображать страницы ошибок для исключений

Если вы хотите показать своим пользователям значимые ошибки вместо простого «извините, что-то пошло не так», у Rails есть хорошая утилита для этой цели.

Откройте файл `app/controllers/application_controller.rb` и вы должны найти что-то вроде этого:



```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
end
```

Теперь мы можем добавить `rescue_from` для восстановления от определенных ошибок:

```
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
```

Рекомендуется не извлекать из `Exception` или `StandardError` иначе Rails не сможет отображать полезные страницы в случае ошибок.

## фильтры

Фильтры - это методы, которые запускаются «до», «после» или «вокруг» действия контроллера. Они наследуются, поэтому, если вы установите их в свой `ApplicationController` они будут выполняться для каждого запроса, получаемого вашим приложением.

### Перед фильтром

Прежде чем фильтры будут выполнены до действия контроллера и могут остановить запрос (и / или перенаправить). Обычно используется проверка, вошел ли пользователь в систему:

```
class ApplicationController < ActionController::Base
  before_action :authenticate_user!

  def authenticate_user!
    redirect_to some_path unless user_signed_in?
  end
end
```

Прежде чем фильтры будут запускаться по запросам до того, как запрос поступит в действие контроллера. Он может вернуть ответ сам и полностью обойти действие.

Другие распространенные применения перед фильтрами проверяют аутентификацию пользователя перед тем, как предоставить им доступ к действию, предназначенному для обработки их запроса. Я также видел, что они использовали для загрузки ресурса из базы данных, проверки разрешений на ресурс или управления переадресацией при других обстоятельствах.

### После фильтра

После того, как фильтры похожи на «раньше», но по мере их запуска после запуска действия они имеют доступ к объекту ответа, который будет отправлен. Итак, коротко после того, как фильтры запускаются после завершения действия. Он может изменить ответ. В большинстве случаев, если что-то происходит в фильтре after, это можно сделать в самом действии, но если есть некоторая логика, которая будет запущена после запуска любого из множества действий, то фильтр после этого является хорошим местом для выполнения Это.

Как правило, я видел фильтры и фильтры, используемые для ведения журнала.

## Фильтр вокруг

Вокруг фильтров может быть логика до и после запуска действия. Он просто уступает действию в любом месте. Обратите внимание, что он не должен уступать действию и может работать без этого, как перед фильтром.

Вокруг фильтров отвечает за выполнение своих связанных действий, уступая, подобно тому, как работают средние стойки.

Вокруг обратных вызовов завершается выполнение действий. Вы можете написать обратный вызов в двух разных стилях. В первом случае обратный вызов представляет собой единый фрагмент кода. Этот код вызывается до того, как действие будет выполнено. Если код обратного вызова вызывает выход, действие выполняется. Когда действие завершается, код обратного вызова продолжает выполняться. Таким образом, код перед выходом выглядит как обратный вызов до действия, а код после выхода - обратный вызов после действия. Если код обратного вызова никогда не вызывает доходность. действие не запускается - это то же самое, что иметь перед обратным вызовом перед возвратом false.

Вот пример фильтра вокруг:

```
around_filter :catch_exceptions

private
  def catch_exceptions
    begin
      yield
    rescue Exception => e
      logger.debug "Caught exception! #{e.message}"
    end
  end
end
```

Это исключает любое действие и помещает сообщение в ваш журнал. Вы можете использовать фильтры для обработки исключений, настройки и удаления, а также множество других случаев.

## Только и кроме

Все фильтры могут применяться к определенным действиям, используя `:only` и `: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
```

## Пропускающий фильтр

Все фильтры (унаследованные) также могут быть пропущены для определенных действий:

```
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
```

Поскольку они унаследованы, фильтры также могут быть определены в «родительском» контроллере `namespace` . Скажем, например, что у вас есть пространство имен для `admin` , и вы, конечно, хотите, чтобы только пользователи-администраторы могли получить к нему доступ. Вы могли бы сделать что-то вроде этого:

```
# 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
```

Помните, что в **Rails 4.x** вы можете использовать `before_filter` вместе с `before_action`, но `before_filter` в настоящее время устарел в **Rails 5.0.0** и будет удален в **5.1**.

## Создание контроллера

Rails предоставляет множество генераторов, конечно же, для контроллеров.

Вы можете создать новый контроллер, выполнив эту команду в папке приложения

```
rails generate controller NAME [action action] [options]
```

*Примечание. Вы также можете использовать псевдонимы `rails g` для вызова `rails generate`*

Например, чтобы создать контроллер для модели `Product`, с действиями `#index` и `#show` вы бы

```
rails generate controller products index show
```

Это создаст контроллер в `app/controllers/products_controller.rb`, причем оба указанных вами действия

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

Он также создаст папку `products` внутри `app/views/`, содержащую два шаблона для действий вашего контроллера (например, `index.html.erb` и `show.html.erb`, обратите внимание, что расширение может варьироваться в зависимости от вашего шаблона, поэтому, если вы Например, при использовании `slim`, генератор создаст `index.html.slim` и `show.html.slim`)

Кроме того, если вы указали какие-либо действия, они также будут добавлены в файл `routes`

```
# config/routes.rb
get 'products/show'
get 'products/index'
```

Rails создает вспомогательный файл для вас, в `app/helpers/products_helper.rb`, а также файлы ресурсов в `app/assets/javascripts/products.js` и `app/assets/stylesheets/products.css`. Что касается представлений, генератор изменяет это поведение в соответствии с тем, что

указано в вашем Gemfile : Т. Gemfile Если вы используете Coffeescript И Sass в своем приложении, генератор контроллера будет вместо этого генерировать products.coffee И products.sass .

Наконец, но не в последнюю очередь, Rails также создает тестовые файлы для вашего контроллера, вашего помощника и ваших просмотров.

Если вы не хотите, чтобы какой-либо из них был создан, вы можете сказать, что Rails пропустили их, просто добавьте любую опцию с помощью

--no- или --skip , вот так:

```
rails generate controller products index show --no-assets --no-helper
```

И генератор пропустит оба assets и helper

Если вам нужно создать контроллер для определенного namespace добавьте его перед NAME :

```
rails generate controller admin/products
```

Это создаст ваш контроллер внутри app/controllers/admin/products\_controller.rb

Rails также может создать для вас полный контроллер RESTful:

```
rails generate scaffold_controller MODEL_NAME # available from Rails 4
rails generate scaffold_controller Product
```

## Rescuing ActiveRecord :: RecordNotFound с redirect\_to

Вы можете спасти исключение RecordNotFound с помощью перенаправления вместо отображения страницы с ошибкой:

```
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
```

Прочитайте ActionController онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2838/actioncontroller>

---

## глава 4: ActionMailer

### Вступление

Action Mailer позволяет отправлять электронные письма из вашего приложения, используя классы и представления почтовой программы. Mailers работают очень похоже на контроллеры. Они наследуют ActionMailer :: Base и живут в приложениях / почтовых программах, и у них есть связанные представления, которые появляются в приложениях / представлениях.

### замечания

Целесообразно обрабатывать отправку электронной почты асинхронно, чтобы не связывать ваш веб-сервер. Это можно сделать с помощью различных сервисов, таких как `delayed_job`.

### Examples

#### Основной Mailer

В этом примере используются четыре разных файла:

- Модель пользователя
- Пользовательская почтовая программа
- Шаблон html для электронной почты
- Шаблон текстового текста для электронной почты

В этом случае модель пользователя называет `approved` метод в `post` отправителе и передает утвержденную `post` (`approved` метод в модели может быть вызван обратным вызовом, методом контроллера и т. Д.). Затем почтовая программа генерирует электронное письмо из шаблона html или plain-text, используя информацию из отправленного `post` (например, заголовок). По умолчанию почтовая программа использует шаблон с тем же именем, что и метод в почтовом ящике (поэтому и метод почтовой программы, и шаблоны имеют имя «одобрено»).

---

## user\_mailer.rb

```
class UserMailer < ActionMailer::Base
  default from: "donotreply@example.com"

  def approved(post)
    @title = post.title
  end
end
```

```
@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!
```

---

## Создание новой почтовой программы

Чтобы создать новую почтовую программу, введите следующую команду

```
rails generate mailer PostMailer
```

Это создаст пустой файл шаблона в `app/mailers/post_mailer.rb` именем *PostMailer*

```
class PostMailer < ApplicationMailer
end
```

Для представления электронной почты будут созданы два файла макета, один для формата html и один для текстового формата.

Если вы предпочитаете не использовать генератор, вы можете создать свои собственные почтовые программы. Убедитесь, что они наследуются от `ActionMailer::Base`

## Добавление вложений

`ActionMailer` также позволяет прикреплять файлы.

```
attachments['filename.jpg'] = File.read('/path/to/filename.jpg')
```

По умолчанию вложения будут закодированы с помощью `Base64`. Чтобы изменить это, вы можете добавить хэш к методу вложений.

```
attachments['filename.jpg'] = {
  mime_type: 'application/gzip',
  encoding: 'SpecialEncoding',
  content: encoded_content
}
```

Вы также можете добавить встроенные вложения

```
attachments.inline['image.jpg'] = File.read('/path/to/image.jpg')
```

## Обратные вызовы ActionMailer

`ActionMailer` поддерживает три обратных вызова

- `before_action`
- `after_action`
- `around_action`

Предоставьте их в своем классе `Mailer`

```
class UserMailer < ApplicationMailer
  after_action :set_delivery_options, :prevent_delivery_to_guests, :set_business_headers
```

Затем создайте эти методы под ключевым словом `private`

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

## Создание запланированного информационного бюллетеня



## Создайте модель информационного бюллетеня :

```
rails g model Newsletter name:string email:string

subl app/models/newsletter.rb

validates :name, presence: true
validates :email, presence: true
```

## Создайте контроллер информационного бюллетеня :

```
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
    end
  end

  private

  def set_newsletter
    @newsletter = Newsletter.find(params[:id])
  end

  def newsletter_params
    params.require(:newsletter).permit(:name, :email)
  end
end
```

После этого измените представление **create.html.erb** на имя **new**. Мы преобразуем этот файл в **частичный просмотр**, который будет сохранен в **нижнем колонтитуле** . Имя будет **\_form.html.erb** .

**Измените имя файла:**

**Для того, чтобы:**

**app / views / newsletters / create.html.erb**

**app / views / newsletters / \_form.html.erb**

После этого установите маршруты:

```
subl app/config/routes.rb

resources :newsletters
```

Позже нам нужно установить форму, которую мы будем использовать для сохранения

каждой почты:

```
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 %>
```

И после этого вставьте нижний колонтитул:

```
subl app/views/layouts/_footer.html.erb

<%= render 'newsletters/form' %>
```

Теперь установите - **letter\_opener** - чтобы можно было просматривать электронную почту в браузере по умолчанию вместо отправки. Это означает, что вам не нужно настраивать доставку электронной почты в среду разработки, и вам больше не нужно беспокоиться о том, чтобы случайно отправить тестовое письмо на чужой адрес.

Сначала добавьте драгоценный камень в среду разработки и запустите команду bundle, чтобы установить ее.

```
subl your_project/Gemfile

gem "letter_opener", :group => :development
```

Затем установите способ доставки в среде разработки:

```
subl your_project/app/config/environments/development.rb

config.action_mailer.delivery_method = :letter_opener
```

Теперь создайте **структуру Mailer** для управления всеми почтовыми серверами, которые мы будем работать. В терминале

```
rails generate mailer UserMailer newsletter_mailer
```

И внутри **UserMailer** мы должны создать метод под названием **Newsletter Mailer**, который будет создан, чтобы содержать внутри последнего сообщения блога и будет выпущен с

помощью рейк-действий. Предположим, что у вас была структура блога, созданная ранее.

```
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
```

После этого создайте **шаблон Mailer** :

```
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>
```

Поскольку мы хотим отправить электронное письмо как отдельный процесс, давайте создадим задачу Rake, чтобы отключить электронную почту. Добавьте новый файл с именем `email_tasks.rake` в каталог `lib / tasks` приложения Rails:

```
touch lib/tasks/email_tasks.rake

desc 'weekly newsletter email'
task weekly_newsletter_email: :environment do
  UserMailer.newsletter_mailer.deliver!
end
```

Среда `send_digest_email::` означает загрузку среды Rails перед запуском задачи, поэтому вы можете получить доступ к классам приложений (например, `UserMailer`) внутри задачи.

Теперь выполнение команды `rake` -Т будет отображать вновь созданную задачу `Rake`. Проверьте все, выполнив задачу и проверив, отправлено ли электронное письмо.

Чтобы проверить, работает ли почтовый метод, выполните команду `rake`:

```
rake weekly_newsletter_email
```

На этом этапе у нас есть рабочая задача, которая может быть запланирована с помощью **crontab** . Поэтому мы установим **Whenever Gem**, который используется для обеспечения четкого синтаксиса для написания и развертывания заданий cron.

```
subl your_project/Gemfile

gem 'whenever', require: false
```

После этого запустите следующую команду, чтобы создать исходный файл `config / schedule.rb` для вас (если папка конфигурации уже присутствует в вашем проекте).

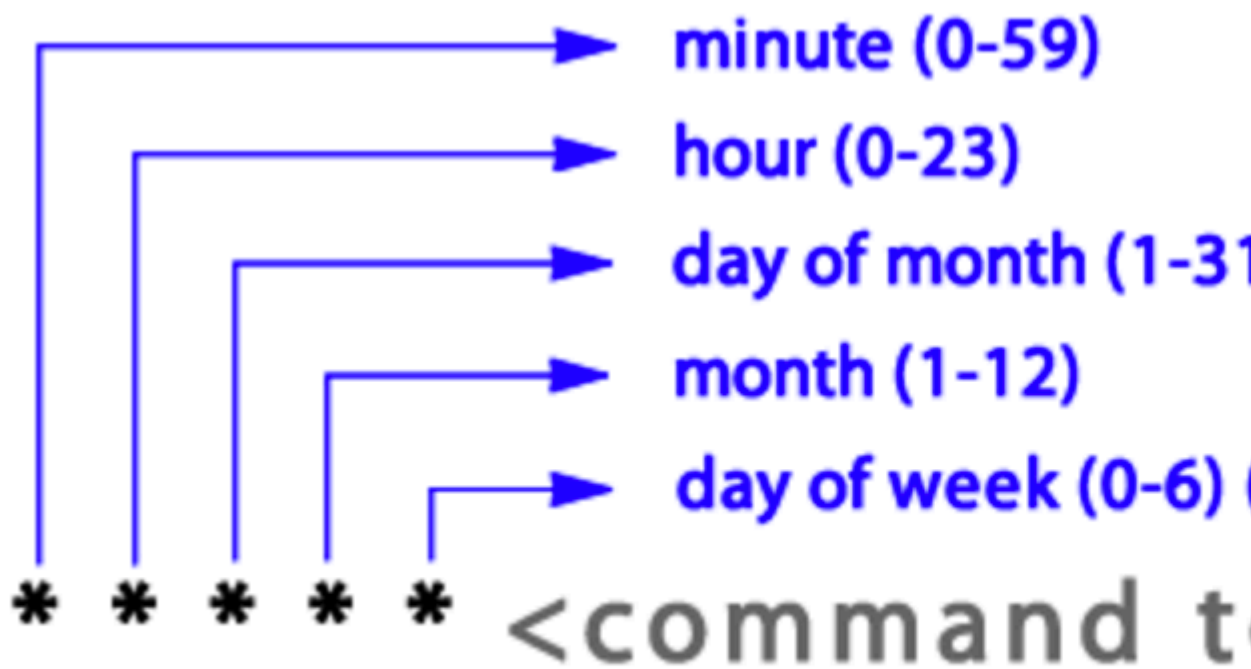
```
wheneverize .

[add] writing `./config/schedule.rb`
[done] wheneverized!
```

Теперь, внутри файла расписания, мы должны создать нашу **CRON JOB** и вызвать метод почтовой программы при определении CRON JOB для управления некоторыми задачами без помощи и в выбранный диапазон времени. Вы можете использовать разные типы синтаксиса, как описано в этой [ссылке](#) .

```
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.m
  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"
end
```

```
every :hour do # Many shortcuts available: :
  runner "SomeModel.ladeeda"
end
```

```
every :sunday, :at => '12pm' do # Use any da
  runner "Task.do_something_great"
end
```

```
every '0 0 27-31 * *' do
  command "echo 'you can use raw cron syntax'"
end
```

```
# run this task only on servers with the :ap
# see Capistrano roles section below
every :day, :at => '12:20am', :roles => [:ap
  rake "app_server:task"
```

был успешно создан, мы можем использовать следующую команду для чтения с терминала, нашу запланированную работу в 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'
```

Теперь, чтобы запустить тест в среде разработки, целесообразно установить следующую строку в главном файле **application.rb**, чтобы приложение могло узнать, какие модели они будут использовать.

```
subl your_project/config/application.rb  
  
config.action_mailer.default_url_options = { :host => "http://localhost:3000/" }
```

Теперь, чтобы **Capistrano V3** сохранил новое задание **Cron** внутри сервера и триггер, который запустил выполнение этой задачи, мы должны добавить следующее требование:

```
subl your_project/Capfile  
  
require 'whenever/capistrano'
```

И вставьте в файл **развертывания** идентификатор, который **CRON JOB** будет использовать об **окружающей среде** и имя приложения .

```
subl your_project/config/deploy.rb  
  
set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

И, после сохранения изменений в каждом файле, запустите команду развертывания capistrano:

```
cap production deploy
```

И теперь ваша JOB была создана и календарна для запуска метода Mailer, который я хочу, и в течение времени, которое мы установили в этих файлах.

## Перехватчик ActionMailer

Action Mailer предоставляет перехватчики в методах перехватчика. Это позволяет вам регистрировать классы, которые вызывают во время цикла доставки почты.

Класс перехватчика должен реализовать метод: `delivering_email (message)`, который будет вызываться до отправки электронной почты, что позволит вам вносить изменения в электронную почту до того, как она попадет в агенты доставки. Ваш класс должен внести любые необходимые изменения непосредственно в переданный экземпляр `Mail :: Message`.

Разработчикам может быть полезно отправить электронную почту самим себе не настоящим пользователям.

Пример регистрации перехватчика actionmailer:

```
# 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
```

Прочитайте ActionMailer онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2481/actionmailer>



---

# глава 5: ActiveJob

## Вступление

Активное задание является основой для объявления заданий и их запуска на различных серверах очереди. Этими заданиями могут быть все, начиная от регулярно запланированных сборов, до выставления счетов, до почтовых рассылок. Все, что можно нарезать на небольшие части работы и работать параллельно, действительно.

## Examples

### Создать работу

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

  def perform(*guests)
    # Do something later
  end
end
```

### Завершить задание

```
# Enqueue a job to be performed as soon as the queuing system is free.
GuestsCleanupJob.perform_later guest
```

Прочитайте ActiveJob онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/8996/activejob>

---

# глава 6: ActiveRecord

## замечания

ActiveModel был создан, чтобы извлечь поведение модели ActiveRecord в отдельную проблему. Это позволяет использовать поведение ActiveModel для любого объекта, а не только для моделей ActiveRecord.

Объекты ActiveRecord включают все это поведение по умолчанию.

## Examples

### Использование ActiveRecord :: Validations

Вы можете проверить любой объект, даже обычный рубин.

```
class User
  include ActiveRecord::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
```

Прочитайте ActiveModel онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/1773/activemodel>

---

# глава 7: ActiveRecord

## Examples

### Создание модели вручную

В то время как использование строительных лесов является быстрым и легким, если вы новичок в Rails или вы создаете новое приложение, позже это может быть полезно просто сделать это самостоятельно, чтобы избежать необходимости проходить через код, созданный эшафотом, чтобы уменьшить его (удалить неиспользованные части и т. д.).

Создание модели может быть таким же простым, как создание файла под `app/models`.

Самая простая модель в ActiveRecord - это класс, расширяющий `ActiveRecord::Base`.

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

Файлы моделей хранятся в `app/models/`, а имя файла соответствует уникальному имени класса:

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

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

Класс наследует все функции ActiveRecord: методы запросов, проверки, обратные вызовы и т. д.

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

Примечание. Убедитесь, что таблица для соответствующей модели существует. Если нет, вы можете создать таблицу, создав [миграцию](#)

Вы можете создать модель и выполнить миграцию по терминалу из следующей команды:

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

и также может назначать внешний ключ (отношение) к модели, следуя команде

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

### Создание модели через генератор

Ruby on Rails предоставляет генератор `model` который можно использовать для создания моделей ActiveRecord. Просто используйте `rails generate model` и укажите название модели.

```
$ rails g model user
```

Помимо файла модели в `app/models`, генератор также создаст:

- Тест в `test/models/user_test.rb`
- Светильники в `test/fixtures/users.yml`
- база данных Миграция в `db/migrate/XXX_create_users.rb`

Вы также можете сгенерировать некоторые поля для модели при ее создании.

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

Это создаст столбцы `email`, `sign_in_count` и день рождения в вашей базе данных с соответствующими типами.

## Создание миграции

# Добавить / удалить поля в существующих таблицах

Создайте миграцию, выполнив:

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

Это создаст миграцию, которая добавит столбец `title` в таблицу `categories`:

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

Аналогично, вы можете создать миграцию, чтобы удалить столбец: `rails generate migration RemoveTitleFromCategories title:string`

Это создаст миграцию, которая удалит столбец `title` из таблицы `categories`:

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

Хотя, строго говоря, указание **типа** ( `:string` в этом случае) **не требуется** для удаления столбца, **это полезно** , поскольку оно предоставляет информацию, необходимую для **ее возврата** .

---

## Создать таблицу

Создайте миграцию, выполнив:

```
rails g CreateUsers name bio
```

Rails распознает намерение создать таблицу из префикса `Create` , остальная часть имени миграции будет использоваться в качестве имени таблицы. В данном примере генерируется следующее:

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

Обратите внимание: команда создания не указала типы столбцов и использовалась `string` по умолчанию.

---

## Создать таблицу соединений

Создайте миграцию, выполнив:

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

Rails обнаруживает намерение создать таблицу объединений путем нахождения `JoinTable` в имени миграции. Все остальное определяется по именам полей, которые вы даете после имени.

```
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
```

Раскомментируйте необходимые `index` операторы и удалите остальные.

# старшинство

Обратите внимание, что `CreateJoinTableParticipation` имя миграции

`CreateJoinTableParticipation` соответствует правилу для создания таблицы: оно имеет префикс `Create` . Но он не создал простую `create_table` . Это связано с тем, что генератор миграции ( [исходный код](#) ) использует **первое совпадение** следующего списка:

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

## Введение в обратные вызовы

Обратный вызов - это метод, который вызывается в определенные моменты жизненного цикла объекта (прямо до или после создания, удаления, обновления, проверки, сохранения или загрузки из базы данных).

Например, скажем, у вас есть листинг, срок действия которого истекает через 30 дней со дня создания.

Один из способов сделать это:

```
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
```

Все доступные методы для обратных вызовов заключаются в следующем, в том же порядке, который они вызывают во время работы каждого объекта:

### Создание объекта

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

- `after_commit / after_rollback`

## Обновление объекта

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

## Уничтожение объекта

- `before_destroy`
- `around_destroy`
- `after_destroy`
- `after_commit / after_rollback`

**ПРИМЕЧАНИЕ:** `after_save` запускается как при создании, так и при обновлении, но всегда после более конкретных обратных вызовов `after_create` и `after_update`, независимо от порядка выполнения макросов.

## Создание таблицы соединений с использованием миграции

Специально полезный для отношения `has_and_belongs_to_many`, вы можете вручную создать таблицу соединений, используя метод `create_table`. Предположим, у вас есть две модели `Tags` и `Projects`, и вы хотите связать их с помощью отношения `has_and_belongs_to_many`. Для объединения экземпляров обоих классов вам нужна таблица соединений.

```
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
```

Фактическое имя таблицы должно следовать этому соглашению: сначала должна идти модель, которая в алфавитном порядке предшествует другой. `Project` предшествует `Tags`, поэтому имя таблицы - `project_tags`.

Кроме того, поскольку целью этой таблицы является маршрутизация связи между экземплярами двух моделей, фактический идентификатор каждой записи в этой таблице не требуется. Вы указываете это, передавая `id: false`

Наконец, как и соглашение в Rails, имя таблицы должно быть составной множественной формой отдельных моделей, но столбец таблицы должен быть в единственном числе.

## Ручное тестирование моделей

Тестирование моделей Active Record через интерфейс командной строки прост. Перейдите в каталог приложения в своем терминале и введите `rails console` чтобы запустить консоль Rails. Отсюда вы можете запускать активные методы записи в своей базе данных.

Например, если у вас была схема базы данных с таблицей `Users`, имеющей `name:string` `column` и `email:string`, вы можете запустить:

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

Затем, чтобы показать эту запись, вы можете запустить:

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

Или, если это ваша первая или единственная запись, вы можете просто получить первую запись, запустив ее:

```
User.first
```

## Использование экземпляра модели для обновления строки

Допустим, у вас есть модель `User`

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

Теперь, чтобы обновить `first_name` и `last_name` пользователя с `id = 1`, вы можете написать следующий код.

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

Вызов `update` попытается обновить данные атрибуты в одной транзакции, возвращая `true` если успешно, и `false` если нет.

Прочитайте [ActiveRecord онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/828/activerecord): <https://riptutorial.com/ru/ruby-on-rails/topic/828/activerecord>



---

# глава 8: ActiveSupport

## замечания

ActiveSupport - это утилита универсальных инструментов, используемых остальной частью Rails.

Одним из основных способов предоставления этих инструментов является monkeypatching родные типы Ruby. Они называются **Core Extensions** .

## Examples

### Основные расширения: доступ к строкам

---

## Строка # в

Возвращает подстроку строкового объекта. Тот же интерфейс, что и `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)  # => ""
```

---

## Строка # из

Возвращает подстроку из заданной позиции в конец строки.

```
str = "hello"
str.from(0)   # => "hello"
str.from(3)   # => "lo"
str.from(-2)  # => "lo"
```

---

## Строка # для

Возвращает подстроку от начала строки до заданной позиции.

Если позиция отрицательная, она отсчитывается от конца строки.

```
str = "hello"
```

```
str.to(0) # => "h"
str.to(3) # => "hell"
str.to(-2) # => "hell"
```

from и to МОЖЕТ ИСПОЛЬЗОВАТЬСЯ В ТАНДЕМЕ.

```
str = "hello"
str.from(0).to(-1) # => "hello"
str.from(1).to(-2) # => "ell"
```

---

## Строка # первый

Возвращает первый символ или заданное количество символов до длины строки.

```
str = "hello"
str.first # => "h"
str.first(1) # => "h"
str.first(2) # => "he"
str.first(0) # => ""
str.first(6) # => "hello"
```

---

## Строка # последний

Возвращает последний символ или заданное количество символов с конца строки, отсчитывающей назад.

```
str = "hello"
str.last # => "o"
str.last(1) # => "o"
str.last(2) # => "lo"
str.last(0) # => ""
str.last(6) # => "hello"
```

Расширения ядра: преобразование строк и даты / времени

---

## Строка # TO\_TIME

Преобразует строку в значение времени. Параметр `form` может быть `:utc` или `:local`, по умолчанию - `: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
```

---

## Строка # to\_date

Преобразует строку в значение Date.

```
"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
```

---

## Строка # to\_datetime

Преобразует строку в значение DateTime.

```
"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
```

Основные расширения: исключение строк

---

## Строка # исключить?

Обратный String#include?

```
"hello".exclude? "lo" # => false
"hello".exclude? "ol" # => true
"hello".exclude? ?h # => false
```

Основные расширения: строковые фильтры

---

## Строка # мармелад

Возвращает версию данной строки без начального или конечного пробела и объединяет все последовательные пробелы в интерьере в одиночные пробелы. Деструктивная версия `squish!` действует непосредственно на экземпляре строки.

Обрабатывает как пробелы ASCII, так и Unicode.

```
%{ Multi-line
  string }.squish # => "Multi-line string"
" foo bar \n \t boo".squish # => "foo bar boo"
```

## Строка # удалить

Возвращает новую строку с удалением всех вхождений шаблонов. Уничтожьте версию `remove!` действует непосредственно на заданную строку.

```
str = "foo bar test"
str.remove(" test")           # => "foo bar"
str.remove(" test", /bar/)   # => "foo "
```

---

## Строка # усечение

Возвращает копию заданной строки, усеченной с заданной длиной, если длина строки больше длины.

```
'Once upon a time in a world far far away'.truncate(27)
# => "Once upon a time in a wo..."
```

Передача строки или регулярного выражения `:separator` для усечения при естественном разрыве

```
'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..."
```

---

## Строка # `truncate_words`

Возвращает строку, усеченную после заданного количества слов.

```
'Once upon a time in a world far far away'.truncate_words(4)
# => "Once upon a time..."
```

Передайте строку или регехр, чтобы указать другой разделитель слов

```
'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..."
```

Последние символы будут заменены строкой `:omission` (по умолчанию «...»)

```
'And they found that many people were sleeping better.'.truncate_words(5, omission: '... (continued)')
# => "And they found that many... (continued)"
```

# Строка # strip\_heredoc

Вырезание полос в heredocs. Ищет наименее исписанную непустую строку и удаляет это количество ведущих пробелов.

```
if options[:usage]
  puts <<-USAGE.strip_heredoc
  This command does such and such.

  Supported options are:
  -h          This message
  ...
  USAGE
end
```

пользователь увидит

```
This command does such and such.

Supported options are:
-h          This message
...
```

Основные расширения: String Inflection

---

# Строка # множественное число

Возвраты множественной формы строки. Необязательно принимает параметр `count` и возвращает особую форму, если `count == 1`. Также принимает параметр `locale` для специфического для языка плюрализации.

```
'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"
```

---

# Строка # образовывать форму единственного числа

Возвращает единственную форму строки. Принимает необязательный параметр `locale` .

```
'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"
```

---

## Строка # constantize

Пытается найти объявленную константу с именем, указанным в строке. Он вызывает `NameError` когда имя не находится в `CamelCase` или не инициализировано.

```
'Module'.constantize # => Module
'Class'.constantize  # => Class
'blargle'.constantize # => NameError: wrong constant name blargle
```

---

## Строка # safe\_constantize

Выполняет `constantize` но возвращает `nil` вместо повышения `NameError` .

```
'Module'.safe_constantize # => Module
'Class'.safe_constantize  # => Class
'blargle'.safe_constantize # => nil
```

---

## Строка # camelize

Преобразует строки в `UpperCamelCase` по умолчанию, если `:lower` задается как параметр преобразует вместо `lowerCamelCase`.

псевдоним: `camelcase`

**Примечание:** также будет конвертировать `/` в `::` что полезно для преобразования путей в пространства имен.

```
'active_record'.camelize      # => "ActiveRecord"
'active_record'.camelize(:lower) # => "activeRecord"
'active_record/errors'.camelize # => "ActiveRecord::Errors"
'active_record/errors'.camelize(:lower) # => "activeRecord::Errors"
```

---

## Строка # titleize

Капитализует все слова и заменяет некоторые символы в строке, чтобы создать более привлекательный заголовок.

Псевдоним: `titlecase`

```
'man from the boondocks'.titleize # => "Man From The Boondocks"  
'x-men: the last stand'.titleize # => "X Men: The Last Stand"
```

---

## Строка # подчеркивание

Делает подчеркнутую, строчную форму из выражения в строке. Реверс `camelize`.

**Примечание.** `underscore` также изменит `::` на `/` чтобы преобразовать пространства имен в пути.

```
'ActiveModel'.underscore # => "active_model"  
'ActiveModel::Errors'.underscore # => "active_model/errors"
```

---

## Строка # dasherize

Заменяет символы подчеркивания тире в строке.

```
'puni_puni'.dasherize # => "puni-puni"
```

---

## Строка # demodulize

Удаляет часть модуля из константного выражения в строке.

```
'ActiveRecord::CoreExtensions::String::Inflections'.demodulize # => "Inflections"  
'Inflections'.demodulize # => "Inflections"  
::Inflections'.demodulize # => "Inflections"  
''.demodulize # => ''
```

---

## Строка # deconstantize

Удаляет самый правый сегмент из константного выражения в строке.

```
'Net::HTTP'.deconstantize # => "Net"  
::Net::HTTP'.deconstantize # => "::Net"  
'String'.deconstantize # => ""  
::String'.deconstantize # => ""  
''.deconstantize # => ""
```

---

## Строка # Параметрирование

Заменяет специальные символы в строке, чтобы ее можно было использовать как часть «симпатичного» URL-адреса.

```
"Donald E. Knuth".parameterize # => "donald-e-knuth"
```

Сохраните случай символов в строке с аргументом `:preserve_case`.

```
"Donald E. Knuth".parameterize(preserve_case: true) # => "Donald-E-Knuth"
```

Очень распространенным вариантом использования для `parameterize` является переопределение метода `to_param` модели ActiveRecord для поддержки более описательных URL-адресов.

```
class Person < ActiveRecord::Base
  def to_param
    "#{id}-#{name.parameterize}"
  end
end

Person.find(1).to_param # => "1-donald-e-knuth"
```

---

## Строка # tableize

Создает имя таблицы, например Rails, для моделей с именами таблиц. Плюрализует последнее слово в строке.

```
'RawScaledScorer'.tableize # => "raw_scaled_scorers"
'ham_and_egg'.tableize    # => "ham_and_eggs"
'fancyCategory'.tableize  # => "fancy_categories"
```

---

## Строка # классифицируют

Возвращает строку имени класса из множественного имени таблицы, такого как Rails для имен таблиц для моделей.

```
'ham_and_eggs'.classify # => "HamAndEgg"
'posts'.classify        # => "Post"
```

---

## Строка # Humanize



Заглавное слово первого слова, превращает подчеркивания в пробелы и разбивает конечный `_id` если он присутствует.

```
'employee_salary'.humanize      # => "Employee salary"
'author_id'.humanize            # => "Author"
'author_id'.humanize(capitalize: false) # => "author"
'_id'.humanize                  # => "Id"
```

---

## Строка # `upcase_first`

Преобразует только первый символ в верхний регистр.

```
'what a Lovely Day'.upcase_first # => "What a Lovely Day"
'w'.upcase_first                 # => "W"
''.upcase_first                  # => ""
```

---

## Строка # `foreign_key`

Создает имя внешнего ключа из имени класса. Передайте `false` параметр, чтобы отключить добавление `_` между именем и `id`.

```
'Message'.foreign_key          # => "message_id"
'Message'.foreign_key(false)  # => "messageid"
'Admin::Post'.foreign_key     # => "post_id"
```

Прочитайте [ActiveSupport онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/4490/activesupport): <https://riptutorial.com/ru/ruby-on-rails/topic/4490/activesupport>

---

# глава 9: API Rails

## Examples

### Создание API-приложения

Чтобы создать приложение Rails, которое будет сервером API, вы можете начать с более ограниченного подмножества Rails в Rails 5.

Чтобы создать новое приложение API Rails:

```
rails new my_api --api
```

Что `--api` - это удалить функциональность, которая не нужна при создании API. Сюда входят сеансы, файлы cookie, активы и все, что делает Rails работать в браузере.

Он также будет настраивать генераторы, чтобы они не генерировали представления, помощники и активы при создании нового ресурса.

Когда вы сравниваете `ApplicationController` с веб-приложением по сравнению с API-приложением, вы увидите, что веб-версия простирается от `ActionController::Base`, тогда как версия API распространяется от `ActionController::API`, которая включает гораздо меньший набор функциональных возможностей.

Прочитайте API Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/4305/api-rails>

# глава 10: Elasticsearch

## Examples

### Установка и тестирование

Первое, что вы хотите сделать для локального развития, - это установить Elasticsearch на свой компьютер и протестировать его, чтобы убедиться, что он работает. Для этого требуется установка Java. Установка довольно проста:

- **Mac OS X:** `brew install elasticsearch`
- **Ubuntu:** `sudo apt-get install elasticsearch`

Затем запустите его:

- **Mac OS X:** `brew services start elasticsearch`
- **Ubuntu:** `sudo service elasticsearch start`

Для его тестирования самый простой способ - `curl`. Для начала может потребоваться несколько секунд, поэтому не паникуйте, если вы сначала не получите ответа.

```
curl localhost:9200
```

Пример ответа:

```
{
  "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" : "You Know, for Search"
}
```

### Настройка инструментов для разработки

Когда вы начинаете работу с Elasticsearch (ES), может быть полезно иметь графический инструмент, который поможет вам изучить ваши данные. Плагин под названием [elasticsearch-head](#) делает именно это. Чтобы установить его, выполните следующие действия:

- Узнайте, в какой папке ES установлена: `ls -l $(which elasticsearch)`
- `cd` в эту папку и запустить двоичный файл плагина: `elasticsearch/bin/plugin -install mobz/elasticsearch-head`

- Откройте `http://localhost:9200/_plugin/head/` в вашем браузере.

Если бы все работало так, как ожидалось, вы должны увидеть хороший графический интерфейс, где вы сможете исследовать свои данные.

## Вступление

У ElasticSearch есть хорошо документированный API JSON, но вы, вероятно, захотите использовать некоторые библиотеки, которые обрабатывают это для вас:

- [Elasticsearch](#) - официальная оболочка низкого уровня для HTTP API
- [Elasticsearch-rails](#) - официальная интеграция Rails на высоком уровне, которая помогает вам подключать ваши модели Rails с помощью ElasticSearch с использованием шаблона ActiveRecord или Repository
- [Chewy](#) - альтернативная, не официальная высокоуровневая интеграция Rails, которая очень популярна и, возможно, имеет лучшую документацию

Давайте используем первый вариант для проверки соединения:

```
gem install elasticsearch
```

Затем запустите рубиновый терминал и попробуйте:

```
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'
```

## Searchkick

Если вы хотите быстро настроить elasticsearch, вы можете использовать камень поиска:

```
gem 'searchkick'
```

Добавьте поисковый запрос к моделям, которые вы хотите найти.

```
class Product < ActiveRecord::Base
  searchkick
end
```

Добавьте данные в индекс поиска.

```
Product.reindex
```

А для запроса используйте:

```
products = Product.search "apples"  
products.each do |product|  
  puts product.name  
end
```

Довольно быстро, знание elasticsearch не требуется ;-)

Дополнительная информация здесь: <https://github.com/ankane/searchkick>

Прочитайте Elasticsearch онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/6500/elasticsearch>

# глава 11: I18n - Интернационализация

## Синтаксис

- `I18n.t ("ключ")`
- `I18n.translate («ключ») # эквивалентно I18n.t ("key")`
- `I18n.t («ключ», число: 4)`
- `I18n.t («ключ», param1: «Что-то», param2: «Else»)`
- `I18n.t («doesnt_exist», по умолчанию: «key») # указать значение по умолчанию, если отсутствует ключ`
- `I18n.locale # =>: ru`
- `I18n.locale =: ru`
- `I18n.default_locale # =>: ru`
- `I18n.default_locale =: ru`
- `t (". key") # то же, что и I18n.t ("key") , но в область действия / шаблон, из которого она I18n.t ("key")`

## Examples

### Использовать I18n в представлении

Предполагая, что у вас есть этот файл локали YAML:

```
# config/locales/en.yml
en:
  header:
    title: "My header title"
```

и вы хотите отобразить строку заголовка, вы можете сделать это

```
# in ERB files
<%= t('header.title') %>

# in SLIM files
= t('header.title')
```

### I18n с аргументами

Параметры `I18n.t` можно передать:

```
# 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) %>

```

И получите следующий результат:

```
"12 users currently online"
```

## плюрализация

Вы можете позволить **I18n** обрабатывать плюрализацию для вас, просто используйте аргумент `count`.

Вам необходимо настроить файл локали следующим образом:

```

# config/locales/en.yml
en:
  online_users:
    one: "1 user is online"
    other: "%{count} users are online"

```

И затем используйте только что созданный ключ, передав аргумент `count` аргументу `I18n.t`:

```

I18n.t("online_users", count: 1)
#=> "1 user is online"

I18n.t("online_users", count: 4)
#=> "4 users are online"

```

## Установить локаль через запросы

В большинстве случаев вы можете установить локаль `i18n`. Возможно, вам захочется установить локаль для текущего сеанса, текущего пользователя или на основе параметра URL. Это легко достижимо путем реализации `before_action` в одном из ваших контроллеров или в `ApplicationController` чтобы иметь его во всех ваших контроллерах.

```

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

# 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
end

```

## URL на основе

Параметр `locale` может исходить из URL-адреса, подобного этому

```
http://yourapplication.com/products?locale=en
```

Или же

```
http://yourapplication.com/en/products
```

Чтобы достичь последнего, вам необходимо изменить `routes`, добавив `scope`:

```

# config/routes.rb
scope "(:locale)", locale: /en|fr/ do
  resources :products
end

```

Поступая таким образом, посещение `http://yourapplication.com/en/products` установит ваш локаль в `:en`. Вместо этого, посетив `http://yourapplication.com/fr/products` он установит его `:fr`. Кроме того, вы не получите ошибку маршрутизации при отсутствии параметра `:locale`, так как посещение `http://yourapplication.com/products` будет загружать локаль `I18n` по умолчанию.

## Основанный на сеансах или основанный на постоянстве



Это предполагает, что пользователь может щелкнуть по значку кнопки / языка, чтобы изменить язык. Действие может перенаправляться на контроллер, который устанавливает сеанс на текущий язык (и, в случае необходимости, сохраняет изменения в базе данных, если пользователь подключен)

```
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
end
```

Вы должны определить `sanitize_language_param` со списком доступных языков и, в конечном счете, обработать ошибки, если язык не существует.

Если у вас очень мало языков, возможно, стоит их определить следующим образом:

```
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?
  end
end
```

*Примечание.* Не забудьте добавить некоторые маршруты к действиям `change_language`

## Default Locale

Помните, что вам нужно установить языковой стандарт по умолчанию для вашего приложения. Вы можете сделать это, установив его в `config/application.rb` :

```
config.i18n.default_locale = :de
```

или путем создания инициализатора в папке `config/initializers` :

```
# config/initializers/locale.rb
I18n.default_locale = :it
```

## Получить локаль из HTTP-запроса

Иногда бывает полезно установить локализацию вашего приложения на основе IP-адреса запроса. Вы можете легко достичь этого с помощью `Geocoder` . Среди многих вещей, `Geocoder` делает `Geocoder` , он также может указать `location request` .

Во-первых, добавьте `Geocoder` в свой `Gemfile`

```
# Gemfile
gem 'geocoder'
```

`Geocoder` добавляет методы `location` и `safe_location` в стандартный объект `Rack::Request` чтобы вы могли легко найти местоположение любого HTTP-запроса по IP-адресу. Эти методы можно использовать в `before_action` в `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
end
```

Помните, что это не будет работать в средах `development` и `test` , поскольку такие вещи, как `0.0.0.0` и `localhost` являются действительными действительными IP-адресами в Интернете.

---

## Ограничения и альтернативы

`Geocoder` очень мощный и гибкий, но его необходимо настроить для работы с *сервисом геокодирования* (см. [Подробнее](#) ); многие из которых ограничивают использование. Также стоит иметь в виду, что вызов внешней службы по каждому запросу может повлиять на производительность.

Для решения этих проблем также стоит рассмотреть:

# 1. Автономное решение

Использование драгоценного камня, такого как `geotip` (см. [Здесь](#)), позволяет выполнять поиск по локальному файлу данных. Там может быть компромисс с точки зрения точности, так как эти файлы данных должны быть обновлены.

## 2. Используйте CloudFlare

Страницы, обслуживаемые CloudFlare, имеют возможность геокодирования прозрачно, причем код страны добавляется в заголовок (`HTTP_CF_IPCOUNTRY`). Более подробную информацию можно найти [здесь](#).

### Перевод атрибутов модели ActiveRecord

`globalize` gem - отличное решение для добавления переводов в ваши модели ActiveRecord. Вы можете установить его, добавив это в свой Gemfile:

```
gem 'globalize', '~> 5.0.0'
```

Если вы используете Rails 5 вам также нужно будет добавить `activemodel-serializers-xml`

```
gem 'activemodel-serializers-xml'
```

Моделирование переводов позволяет вам переводить значения атрибутов ваших моделей, например:

```
class Post < ActiveRecord::Base
  translates :title, :text
end

I18n.locale = :en
post.title # => Globalize rocks!

I18n.locale = :he
post.title # => טלוש זיילאבולג!
```

После того, как вы определили свои атрибуты модели, которые нужно перевести, вам нужно создать таблицу переводов через миграцию. `globalize` предоставляет `create_translation_table!` и `drop_translation_table!`,

Для этой миграции вам нужно использовать `up` и `down`, а не `change`. Кроме того, чтобы успешно выполнить эту миграцию, сначала необходимо определить переведенные атрибуты в своей модели, как показано выше. Правильная миграция для предыдущей модели `Post`:

```
class CreatePostsTranslationTable < ActiveRecord::Migration
```

```

def up
  Post.create_translation_table! title: :string, text: :text
end

def down
  Post.drop_translation_table!
end
end

```

Вы также можете передавать параметры для определенных параметров, например:

```

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

```

Если у вас уже есть какие-либо **существующие данные** в ваших столбцах перевода, вы можете легко перенести их в таблицу переводов, регулируя миграцию:

```

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

```

**Убедитесь, что вы удалили переведенные столбцы из родительской таблицы после того, как все ваши данные безопасно перенесены.** Чтобы автоматически удалить переведенные столбцы из родительской таблицы после переноса данных, добавьте параметр `remove_source_columns` в перенос:

```

class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table!({
      title: :string,
      text: :text
    }, {
      migrate_data: true,
      remove_source_columns: true
    })
  end
end

```

```
end

def down
  Post.drop_translation_table! migrate_data: true
end
end
```

Вы также можете добавить новые поля в ранее созданную таблицу переводов:

```
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
```

## Использовать I18n с HTML-тегами и символами

```
# config/locales/en.yml
en:
  stackoverflow:
    header:
      title_html: "Use <strong>I18n</strong> with Tags & Symbols"
```

Обратите внимание на добавление дополнительного `_html` после названия `title`.

И в Представлениях,

```
# ERB
<h2><%= t(:title_html, scope: [:stackoverflow, :header]) %></h2>
```

Прочитайте I18n - Интернационализация онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2772/i18n---интернационализация>

---

# глава 12: Mongoid

## Examples

### Монтаж

Сначала добавьте `Mongoid` в ваш `Gemfile` :

```
gem "mongoid", "~> 4.0.0"
```

а затем запустить `bundle install` . Или просто запустите:

```
$ gem install mongoid
```

После установки запустите генератор для создания файла конфигурации:

```
$ rails g mongoid:config
```

который создаст файл `(myapp)/config/mongoid.yml` .

### Создание модели

Создайте модель (позвоните на нее `User` ), запустив:

```
$ rails g model User
```

который будет генерировать файл `app/models/user.rb` :

```
class User
  include Mongoid::Document

end
```

Это все, что вам нужно, чтобы иметь модель (хотя и не что иное, как поле `id` ). В отличие от `ActiveRecord` нет файлов миграции. Вся информация о базе данных для модели содержится в файле модели.

Временные метки автоматически не включаются в вашу модель при ее создании. Чтобы добавить `created_at` и `updated_at` в вашу модель, добавьте

```
include Mongoid::Timestamps
```

к вашей модели внизу `include Mongoid::Document` :

```
class User
  include Mongoid::Document
  include Mongoid::Timestamps
end
```

## ПОЛЯ

В соответствии с [Монгольской документацией](#) существует 16 допустимых типов полей:

- массив
- BigDecimal
- логический
- Дата
- DateTime
- терка
- гашиш
- целое число
- BSON :: ObjectId
- BSON :: Binary
- Спектр
- Regexp
- строка
- Условное обозначение
- Время
- TimeWithZone

Чтобы добавить поле (назовем его `name` и пусть оно будет `String`), добавьте его в файл модели:

```
field :name, type: String
```

Чтобы установить значение по умолчанию, просто перейдите по `default` :

```
field :name, type: String, default: ""
```

## Классические ассоциации

Mongoid позволяет классические ассоциации ActiveRecord :

- Один-к-одному: `has_one / belongs_to`
- Один-ко-многим: `has_many / belongs_to`
- Много-ко-многим: `has_and_belongs_to_many`

Чтобы добавить ассоциацию (скажем, сообщения пользователя `has_many`), вы можете добавить это в свой файл модели `User` :

```
has_many :posts
```

и это для вашего файла модели `Post` :

```
belongs_to :user
```

Это добавит поле `user_id` в вашу модель `Post` , добавит `user` метод в класс `Post` и добавит метод `posts` в ваш класс `User` .

## Встраиваемые ассоциации

Монгоид допускает встраиваемые ассоциации:

- `embeds_one` : `embeds_one` / `embedded_in`
- **Один-ко-многим:** `embeds_many` / `embedded_in`

Чтобы добавить ассоциацию (допустим, пользователь `embeds_many` адресов), добавьте это в свой файл `User` :

```
embeds_many :addresses
```

и это в файл модели вашего `Address` :

```
embedded_in :user
```

Это позволит вставить `Address` в модель `User` , добавив метод `addresses` в ваш класс `User` .

## Вызовы базы данных

Mongoid пытается иметь аналогичный синтаксис для `ActiveRecord` когда это возможно. Он поддерживает эти вызовы (и многие другие)

```
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
```



Прочитайте Mongoid онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3071/mongoid>

---

# глава 13: Rails Cookbook - расширенные рецепты рельсов / методы обучения и кодирования

## Examples

### Игра с таблицами с использованием консоли rails

---

#### Просмотр таблиц

```
ActiveRecord::Base.connection.tables
```

#### Удалите любую таблицу .

```
ActiveRecord::Base.connection.drop_table("users")
-----OR-----
ActiveRecord::Migration.drop_table(:users)
-----OR-----
ActiveRecord::Base.connection.execute("drop table users")
```

#### Удалить индекс из существующего столбца

```
ActiveRecord::Migration.remove_index(:users, :name => 'index_users_on_country')
```

где `country` - это имя столбца в файле миграции с уже добавленным индексом в таблице `users` как показано ниже: -

```
t.string :country, add_index: true
```

#### Удалить ограничение внешнего ключа

```
ActiveRecord::Base.connection.remove_foreign_key('food_items', 'menus')
```

Где `menus has_many food_items` и их соответствующие миграции тоже.

#### Добавить столбец

```
ActiveRecord::Migration.remove_column :table_name, :column_name
```

например:-

```
ActiveRecord::Migration.add_column :profiles, :profile_likes, :integer, :default => 0
```

## Методы Rails - возврат логических значений

Любой метод в модели Rails может возвращать логическое значение.

### простой метод-

```
##this method return ActiveRecord::Relation
def check_if_user_profile_is_complete
  User.includes( :profile_pictures, :address, :contact_detail).where("user.id = ?",self)
end
```

### Снова простой метод, возвращающий логическое значение -

```
##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
```

Таким образом, тот же метод теперь будет возвращать boolean вместо всего остального :).

## Обработка ошибки - неопределенный метод `где` для #

Иногда мы хотим использовать запрос `where` на коллекцию возвращенных записей, которая не является `ActiveRecord::Relation`. Hence мы получаем указанную выше ошибку, поскольку предложение `Where` известно в `ActiveRecord` а не в `Array`.

Для этого есть точное решение с помощью `Joins`.

### ПРИМЕР : -

Предположим, мне нужно найти все профили пользователей (`UserProfile`), которые являются активными, которые не являются пользователем (пользователем) с `id = 10`.

```
UserProfiles.includes(:user=>:profile_pictures).where(:active=>true).map(&:user).where.not(:id=>10)
```

Таким образом, над запросом будет отказано после `map` как `map` вернет `array` который не будет работать с предложением `where`.

Но с помощью объединений он будет работать,

```
UserProfiles.includes(:user=>:profile_pictures).where(:active=>true).joins(:user).where.not(:id=>10)
```

Поскольку `joins` будут выводить похожие записи, такие как `map` но они будут `ActiveRecord` а

**не** Array .

Прочитайте Rails Cookbook - расширенные рецепты рельсов / методы обучения и кодирования онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/7259/rails-cookbook---расширенные-рецепты-рельсов---методы-обучения-и-кодирования>

---

# глава 14: Rails Engine - модульные рельсы

## Вступление

### Краткий обзор двигателей Rails

Двигатели - это небольшие приложения Rails, которые можно использовать для добавления функциональных возможностей в приложение, в котором они размещены. Класс, определяющий приложение Ruby on Rails, - это `Rails::Application` которое на самом деле наследует много своего поведения от `Rails::Engine`, класса, определяющего движок. Можно сказать, что регулярное приложение Rails - это просто движок с большим количеством функций.

## Синтаксис

- `rails plugin new my_module` - доступный

## Examples

### Создание модульного приложения

# Начиная

Во-первых, давайте создадим новое приложение Ruby on Rails:

```
rails new ModularTodo
```

Следующий шаг - создать движок!

```
cd ModularTodo && rails plugin new todo --mountable
```

Мы также создадим папку «engine» для хранения движков (даже если у нас есть только один!).

```
mkdir engines && mv todo ./engines
```

Двигатели, как и драгоценные камни, поставляются с файлом `gemspec`. Давайте дадим некоторые реальные значения, чтобы избежать предупреждений.

```
#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       = "http://samurails.com"
  s.summary        = "Todo Module"
  s.description    = "Todo Module for Modular Rails article"
  s.license        = "MIT"

  #Moar stuff
  #...
end
```

Теперь нам нужно добавить движок Todo в родительское приложение Gemfile.

```
#ModularTodo/Gemfile
#Other gems
gem 'todo', path: 'engines/todo'
```

Давайте запустим `bundle install`. В списке драгоценных камней вы должны увидеть следующее:

```
Using todo 0.0.1 from source at engines/todo
```

Отлично, наш двигатель Todo загружен правильно! Прежде чем мы начнем кодирование, у нас есть одна последняя вещь: смонтировать движок Todo. Мы можем сделать это в файле `routes.rb` в родительском приложении.

```
Rails.application.routes.draw do
  mount Todo::Engine => "/", as: 'todo'
end
```

Мы монтируем его в `/` но мы также можем сделать его доступным `/todo`. Поскольку мы имеем только один модуль, `/` отлично.

Теперь вы можете запустить свой сервер и проверить его в своем браузере. Вы должны увидеть представление Rails по умолчанию, потому что мы еще не определили какие-либо контроллеры / представления. Давайте сделаем это сейчас!

---

## Создание списка Тодо

Мы собираемся поднять модель под названием `Task` внутри модуля Todo, но для

правильной миграции базы данных из родительского приложения нам нужно добавить небольшой инициализатор в файл `engine.rb`.

```
#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
```

Вот и все, теперь, когда мы запускаем миграции из родительского приложения, миграция в движке Todo также будет загружена.

Давайте создадим модель `Task`. Команда `scaffold` должна запускаться из папки с двигателем.

```
cd engines/todo && rails g scaffold Task title:string content:text
```

Запустите миграцию из родительской папки:

```
rake db:migrate
```

Теперь нам просто нужно определить корневой путь внутри движка Todo:

```
#ModularTodo/engines/todo/config/routes.rb
Todo::Engine.routes.draw do
  resources :tasks
  root 'tasks#index'
end
```

Вы можете играть с ним, создавать задания, удалять их ... О, подождите, удаление не работает! Зачем?! Ну, похоже, JQuery не загружен, поэтому давайте добавим его в файл `application.js` внутри движка!

```
// ModularTodo/engines/todo/app/assets/javascripts/todo/application.js
//= require jquery
//= require jquery_ujs
//= require_tree .
```

Да, теперь мы можем уничтожить задачи!

Прочитайте Rails Engine - модульные рельсы онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9080/rails-engine---модульные-рельсы>



# глава 15: Rails -Engines

## Вступление

Двигатели можно рассматривать как миниатюрные приложения, которые обеспечивают функциональность для своих хост-приложений. Приложение Rails на самом деле просто движок с наддувом, а класс `Rails :: Application` наследует много своего поведения от `Rails :: Engine`.

Двигатели - многоразовые приложения / плагины рельсов. Он работает как драгоценный камень. Известные двигатели - это устройства, драгоценные камни Spree, которые легко интегрируются с рельсами.

## Синтаксис

- `rails plugin new [engine name] --mountable`

## параметры

параметры	Цель
<code>--mountable</code>	опция сообщает генератору, что вы хотите создать «монтируемый» и изолированный от пространства имен движок
<code>--полный</code>	опция сообщает генератору, что вы хотите создать движок, включая структуру скелета

## замечания

Двигатели - очень хорошие возможности для создания многоразового плагина для применения в рельсах

## Examples

### Известные примеры

Создание простого движка блога

```
rails plugin new [engine name] --mountable
```

Примеры известных двигателей

[Устройство](#) (подлинник для рельсов)

[Spree](#) (Электронная торговля)

Прочитайте [Rails -Engines](#) онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/10881/rails--engines>

# глава 16: Rails logger

## Examples

### Rails.logger

Всегда используйте `Rails.logger.{debug|info|warn|error|fatal}` а не `puts`. Это позволяет вашим журналам вписываться в стандартный формат журнала, иметь отметку времени и уровень, поэтому вы выбираете, достаточно ли они важны для отображения в определенной среде. Вы можете просмотреть отдельные файлы журналов для своего приложения в каталоге `log/` с именем среды приложения rails. например: `development.log` или `production.log` или `staging.log`

Вы можете легко создавать журналы обработки рельсов с помощью LogRotate. Вам просто нужно выполнить небольшую конфигурацию, как показано ниже.

Откройте `/etc/logrotate.conf` с вашим любимым редактором `vim` или `nano` linux и добавьте ниже код в этот файл внизу.

```
/YOUR/RAILSAPP/PATH/log/*.log {
  daily
  missingok
  rotate 7
  compress
  delaycompress
  notifempty
  copytruncate
}
```

Итак, **как это работает** Это фантастически легко. Каждый бит конфигурации выполняет следующие действия:

- **ежедневно** - каждый день поворачивайте файлы журнала. Здесь вы также можете использовать еженедельно или ежемесячно.
- **missingok** - Если файл журнала не существует, игнорируйте его
- **вращать 7** - Сохранять только 7 дней бревна
- **compress** - GZip файл журнала при вращении
- **delaycompress** - повернуть файл один день, а затем сжать его на следующий день, чтобы мы могли быть уверены, что он не будет мешать серверу Rails
- **notifempty** - не вращать файл, если журналы пусты
- **copytruncate** - Скопировать файл журнала, а затем опорожнить его. Это гарантирует, что файл журнала Rails записывается всегда, поэтому вы не получите проблем, потому что файл фактически не изменяется. Если вы не используете это, вам нужно будет перезапустить приложение Rails каждый раз.

**Запуск Logrotate.** Поскольку мы просто написали эту конфигурацию, вы хотите ее протестировать.

Чтобы запустить logrotate вручную, просто выполните: `sudo /usr/sbin/logrotate -f /etc/logrotate.conf`

Вот и все.

Прочитайте Rails logger онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3904/rails-logger>

---

# глава 17: RSpec и Ruby on Rails

## замечания

RSpec - это тестовая среда для Ruby или, как определено официальной документацией, *RSpec - это инструмент, ориентированный на поведение для программистов Ruby*.

В этом разделе описывается использование RSpec с помощью Ruby on Rails. Для получения подробной информации о RSpec см. [Раздел RSpec](#).

## Examples

### Установка RSpec

Если вы хотите использовать RSpec для проекта Rails, вы должны использовать `rspec-rails`, который может автоматически генерировать `rspec-rails` файлы и файлы спецификаций (например, при создании моделей, ресурсов или лесов с использованием `rails generate`).

Добавьте `rspec-rails` в группы `:development` и `:test` в Gemfile:

```
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Запустите `bundle` для установки зависимостей.

Инициализируйте его с помощью:

```
rails generate rspec:install
```

Это создаст `spec/` папку для ваших тестов вместе со следующими конфигурационными файлами:

- `.rspec` содержит параметры по умолчанию для инструмента командной строки командной строки `rspec`
- `spec/spec_helper.rb` содержит основные параметры конфигурации RSpec
- `spec/rails_helper.rb` добавляет дополнительные параметры конфигурации, которые более специфичны для использования RSpec и Rails вместе.

Все эти файлы написаны с разумными настройками по умолчанию, чтобы вы начали, но вы можете добавлять функции и изменять конфигурации в соответствии с вашими потребностями по мере роста набора тестов.

Прочитайте RSpec и Ruby on Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/5335/rspec-и-ruby-on-rails>

---

# глава 18: Turbolinks

## Вступление

Turbolinks - это javascript-библиотека, которая ускоряет навигацию вашего веб-приложения. Когда вы переходите по ссылке, Turbolinks автоматически извлекает страницу, свопирует ее в `<body>` и объединяет ее `<head>`, причем все без затрат на полную загрузку страницы.

## замечания

Как разработчик рельсов, вы, скорее всего, будете взаимодействовать с турбовинтами минимально во время своего развития. Это, однако, важная библиотека, чтобы быть знакомой, потому что это может быть причиной некоторых труднодоступных ошибок.

---

## Ключевые вынос:

- Привязать к `turbolinks:load` событие `turbolinks:load` вместо события `document.ready`
- Используйте атрибут `data-turbolinks=false` чтобы отключить функциональные возможности `turbolink` по каждой ссылке.
- Используйте атрибут `data-turbolinks-permanent` для сохранения элементов в нагрузках на странице и для предотвращения ошибок, связанных с кешем.

Для более глубокого изучения турбонасосов посетите [официальный репозиторий github](#) .

Кредит для большей части этой документации относится к людям, которые разработали документацию по турбовинтовым устройствам в репозитории `github`.

## Examples

### Привязка к концепции `turbolink` загрузки страницы

С `turbolinks` традиционный подход к использованию:

```
$(document).ready(function() {  
  // awesome code  
});
```

не будет работать. При использовании `turbolinks` событие `$(document).ready()` будет срабатывать только один раз: при начальной загрузке страницы. С этого момента каждый раз, когда пользователь нажимает ссылку на вашем веб-сайте, `turbolinks` перехватывает событие щелчка ссылки и делает запрос `ajax` заменять тег `<body>` и объединять теги `<`

head>. Весь процесс запускает понятие «посещения» в судах турбонаводов. Поэтому вместо использования традиционного синтаксиса `document.ready()` выше вам придется привязываться к событию посещения `turboLink` следующим образом:

```
// pure js
document.addEventListener("turbo:load", function() {
  // awesome code
});

// jQuery
$(document).on('turbo:load', function() {
  // your code
});
```

## Отключить турбонауки по определенным ссылкам

Очень просто отключить турбовинты по конкретным ссылкам. Согласно [официальной документации по турбонагнетателям](#) :

TurboLinks можно отключить по каждой ссылке путем аннотации ссылки или любого из ее предков с данными `turboLinks = "false"`.

## Примеры:

```
// 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>
```

## Понимание посещения приложений

Визиты приложений инициируются нажатием на ссылку с поддержкой TurboLinks или программным путем путем вызова

```
TurboLinks.visit(location)
```

По умолчанию функция посещения использует действие «вперед». Понятно, что поведение по умолчанию для функции посещения - это переход на страницу, указанную параметром «location». Всякий раз, когда страница посещается, turboLinks вставляет новую запись в



историю браузера, используя `history.pushState`. История важна, потому что `turbolinks` будет пытаться использовать историю для загрузки страниц из кеша, когда это возможно. Это позволяет очень быстро выполнять рендеринг страниц для часто посещаемых страниц.

Однако, если вы хотите посетить место, не вставляя историю в стек, вы можете использовать действие «replace» для функции посещения так:

```
// using links
<a href="/edit" data-turbolinks-action="replace">Edit</a>

// programatically
Turbolinks.visit("/edit", { action: "replace" })
```

Это заменит верхнюю часть стека истории новой страницей так, чтобы общее количество элементов в стеке оставалось неизменным.

Существует также действие «восстановления», которое помогает в **ВОССТАНОВЛЕНИИ**, посещения, которые происходят в результате нажатия пользователем кнопки «вперед» или «назад» в их браузере. `Turbolinks` обрабатывает эти типы событий внутри себя и рекомендует пользователям не вмешиваться в поведение по умолчанию.

## Отмена посещений до их начала

`Turbolinks` предоставляет прослушатель событий, который может использоваться для остановки посещений. Слушайте `turbolinks:before-visit` событие `turbolinks:before-visit` чтобы получить уведомление о начале своего визита.

В обработчике событий вы можете использовать:

```
// pure javascript
event.data.url
```

или же

```
// jQuery
$event.originalEvent.data.url
```

для получения места посещения. Затем визит можно отменить, позвонив:

```
event.preventDefault()
```

## НОТА:

Согласно [официальным документам turbolinks](#) :

Реставрационные посещения не могут быть отменены и не запускать

турбонауки: перед посещением.

## Сохранение элементов на разных страницах

Рассмотрим следующую ситуацию: представьте, что вы являетесь разработчиком веб-сайта социальных сетей, который позволяет пользователям дружить с другими пользователями и использует турболинки для ускорения загрузки страницы. В правом верхнем углу каждой страницы сайта есть номер, указывающий общее количество друзей, которые пользователь имеет в настоящее время. Представьте, что вы используете свой сайт и у вас есть 3 друга. Когда новый друг добавляется, у вас есть javascript, который запускает, который обновляет счетчик друзей. Представьте, что вы только что добавили нового друга и что ваш javascript правильно работал и обновил счет друга в правом верхнем углу страницы, чтобы теперь сделать 4. Теперь представьте, что вы нажимаете кнопку возврата браузера. Когда страница загружается, вы замечаете, что счетчик друзей говорит 3, даже если у вас четверо друзей.

Это довольно распространенная проблема, и это решение для turbolinks. Причина, по которой возникает проблема, заключается в том, что turbolinks автоматически загружает страницы из кеша, когда пользователь нажимает кнопку «Назад». Кэшированная страница не всегда будет обновляться с помощью базы данных.

Чтобы решить эту проблему, представьте, что вы передаете счет другу внутри тега `<div>` с идентификатором «friend-count»:

```
<div id="friend-count" data-turbolinks-permanent>3 friends</div>
```

Добавляя атрибут `data-turbolinks-permanent`, вы говорите, что turbolinks сохраняет определенные элементы в нагрузках на страницу. [Официальные документы говорят](#) :

Назначьте постоянные элементы, указав им идентификатор HTML и аннотируя их данными-turbolink-постоянными. Перед каждым рендером Turbolinks сопоставляет все постоянные элементы по id и передает их с исходной страницы на новую страницу, сохраняя свои данные и прослушватели событий.

Прочитайте Turbolinks онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9331/turbolinks>

---

# глава 19: Авторизация Rails 5 API

## Examples

### Аутентификация с помощью Rails `authenticate_with_http_token`

```
authenticate_with_http_token do |token, options|  
  @user = User.find_by(auth_token: token)  
end
```

Вы можете проверить эту конечную точку с помощью `curl`, сделав запрос, например

```
curl -IH "Authorization: Token token=my-token" http://localhost:3000
```

Прочитайте Авторизация Rails 5 API онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/7852/авторизация-rails-5-api>

---

# глава 20: Авторизация с помощью CanCan

## Вступление

[CanCan](#) - это простая стратегия авторизации для Rails, которая отделена от пользовательских ролей. Все разрешения хранятся в одном месте.

## замечания

Перед тем, как использовать CanCan, не забудьте создать Пользователей, создав драгоценный камень или вручную. Чтобы получить максимальную функциональность CanCan, создайте пользователя Admin.

## Examples

### Начало работы с CanCan

[CanCan](#) - популярная библиотека авторизации для Ruby on Rails, которая ограничивает доступ пользователей к определенным ресурсам. Последний камень (CanCanCan) является продолжением мертвого проекта [CanCan](#) .

Разрешения определяются в классе « `Ability` » и могут использоваться из контроллеров, представлений, помощников или любого другого места в коде.

Чтобы добавить поддержку авторизации в приложение, добавьте камень `Gemfile` в `Gemfile` :

```
gem 'cancancan'
```

Затем определите класс способности:

```
# app/models/ability.rb
class Ability
  include CanCan::Ability

  def initialize(user)
    end
  end
end
```

Затем проверьте авторизацию с помощью `load_and_authorize_resource` для загрузки разрешенных моделей в контроллер:

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

  def show
```

```
    # @article is already loaded and authorized
  end
end
```

authorize! проверить авторизацию или создать исключение

```
def show
  @article = Article.find(params[:id])
  authorize! :read, @article
end
```

can? чтобы проверить, разрешен ли объект для конкретного действия в любом месте контроллеров, представлений или помощников

```
<% if can? :update, @article %>
  <%= link_to "Edit", edit_article_path(@article) %>
<% end %>
```

**Примечание.** Предполагается, что подписанный пользователь предоставлен методом `current_user`.

## Определение способностей

Способности определяются в классе `Ability` используя методы `can` и `cannot`. Рассмотрим следующий приведенный ниже пример базовой ссылки:

```
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
```

## Обработка большого количества способностей

Как только число определений способностей начинает расти в количестве, становится все труднее обрабатывать файл способности.

Первой стратегией для решения этой проблемы является перемещение способностей в осмысленные методы, как в этом примере:

```
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
```

Как только этот класс станет достаточно большим, вы можете попробовать разбить его на разные классы, чтобы справиться с различными обязанностями:

```
# 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
    end
  end
end
```

```
    end
  else
    self.merge Abilities::Guest.new(user)
  end
end
end
end
```

а затем определите эти классы как:

```
# app/models/abilities/guest.rb
module Abilities
  class Guest
    include CanCan::Ability

    def initialize(user)
      # Abilities for anonymous visitors only
    end
  end
end
```

и т. д. с возможностями `Abilities::Authenticated`, `Abilities::Admin` или любым другим.

## Быстро проверить способность

Если вы хотите быстро протестировать, если класс способности дает правильные разрешения, вы можете инициализировать возможность в консоли или в другом контексте с загруженной средой рельсов, просто передайте экземпляр пользователя для проверки на:

```
test_ability = Ability.new(User.first)
test_ability.can?(:show, Post) #=> true
other_ability = Ability.new(RestrictedUser.first)
other_ability.cannot?(:show, Post) #=> true
```

Дополнительная информация: <https://github.com/ryanb/cancan/wiki/Testing-Abilities>

Прочитайте Авторизация с помощью CanCan онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3021/авторизация-с-помощью-cancan>

---

# глава 21: Активные вакансии

## Examples

### Вступление

Доступно с Rails 4.2, Active Job - это среда для объявления заданий и их запуска на различных серверах очередей. Повторяющиеся или пунктуальные задачи, которые не блокируются и могут выполняться параллельно, являются хорошими вариантами использования для активных заданий.

### Пример задания

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

  def perform(user)
    # this will happen later
    user.unsubscribe
  end
end
```

### Создание активного задания через генератор

```
$ rails g job user_unsubscribe
```

Прочитайте **Активные вакансии онлайн**: <https://riptutorial.com/ru/ruby-on-rails/topic/8033/активные-вакансии>



---

# глава 22: Активные транзакции

## ActiveRecord

### замечания

Транзакции являются защитными блоками, где заявления SQL являются только постоянными, если все они могут быть успешными как одно атомное действие. Классический пример - это передача между двумя учетными записями, где вы можете получить только депозит, если списание отменено, и наоборот. Транзакции обеспечивают целостность базы данных и защищают данные от ошибок программы или разбивки баз данных. Поэтому в основном вы должны использовать блоки транзакций, когда у вас есть несколько операторов, которые должны выполняться вместе или вообще не выполняться.

## Examples

### Основной пример

Например:

```
ActiveRecord::Base.transaction do
  david.withdrawal(100)
  mary.deposit(100)
end
```

Этот пример будет брать деньги только у Дэвида и отдать его Мэри, если ни отход, ни депозит не приведут к исключению. Исключения заставят ROLLBACK вернуть базу данных в состояние до начала транзакции. Однако имейте в виду, что у объектов не будет данных экземпляра, возвращаемых в их предварительное транзакционное состояние.

### Различные классы ActiveRecord в одной транзакции

Хотя метод класса транзакции вызывается в некотором классе ActiveRecord, объекты внутри блока транзакций не обязательно должны быть экземплярами этого класса. Это связано с тем, что транзакции - это подключение по каждой базе данных, а не для каждой модели.

В этом примере запись баланса сохраняется в транзакции, даже если транзакция вызывается в классе Account:

```
Account.transaction do
  balance.save!
  account.save!
end
```

Метод транзакции также доступен как метод экземпляра модели. Например, вы также можете сделать это:

```
balance.transaction do
  balance.save!
  account.save!
end
```

## Несколько соединений с базой данных

Транзакция действует на одно соединение с базой данных. Если у вас несколько баз данных, специфичных для класса, транзакция не будет защищать взаимодействие между ними. Одним из способов является начало транзакции для каждого класса, чьи модели вы изменяете:

```
Student.transaction do
  Course.transaction do
    course.enroll(student)
    student.units += course.units
  end
end
```

Это плохое решение, но полностью распределенные транзакции выходят за рамки ActiveRecord.

## сохранение и уничтожение автоматически завертываются в транзакцию

Оба метода `#save` и `#destroy` завершаются транзакцией, которая гарантирует, что все, что вы делаете при проверке или обратном вызове, произойдет под защищенной оболочкой. Таким образом, вы можете использовать проверки для проверки значений, на которые зависит транзакция, или вы можете `after_*` исключения в обратных `after_*` для отката, включая `after_*` callbacks.

Как следствие, изменения в базе данных не отображаются за пределами вашего соединения до завершения операции. Например, если вы попытаетесь обновить индекс поисковой системы в `after_save` не увидит обновленную запись. `after_commit` вызов `after_commit` является единственным, который запускается после завершения обновления.

## Callbacks

Существует два типа обратных вызовов, связанных с транзакциями и `after_rollback` транзакций: `after_commit` и `after_rollback`.

`after_commit` callbacks `after_commit` для каждой записи, сохраненной или уничтоженной в транзакции сразу после совершения транзакции. `after_rollback` вызовы `after_rollback` вызываются в каждой записи, сохраненной или уничтоженной в транзакции сразу после отката транзакции или точки сохранения.

Эти обратные вызовы полезны для взаимодействия с другими системами, поскольку вам гарантируется, что обратный вызов будет выполняться только в том случае, если база данных находится в постоянном состоянии. Например, `after_commit` - хорошее место, чтобы положить крючок в очистку кэша, поскольку очистка его изнутри транзакции может привести к восстановлению кэша до обновления базы данных.

## Откат транзакции

`ActiveRecord::Base.transaction` использует исключение `ActiveRecord::Rollback` чтобы отличить преднамеренный откат от других исключительных ситуаций. Обычно повышение исключения приводит к `.transaction` метод `.transaction` откатывается от транзакции базы данных и передает исключение. Но если вы `ActiveRecord::Rollback` исключение `ActiveRecord::Rollback`, транзакция базы данных будет отброшена, не передавая исключение.

Например, вы можете сделать это в своем контроллере для отката транзакции:

```
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
    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
```

Прочитайте [Активные транзакции ActiveRecord онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/4688/активные-транзакции-activerecord](https://riptutorial.com/ru/ruby-on-rails/topic/4688/активные-транзакции-activerecord)

---

# глава 23: Активные транзакции ActiveRecord

## Вступление

Транзакции ActiveRecord являются защитными блоками, в которых последовательность активных запросов записи является постоянной, если все они могут выполняться как одно атомное действие.

## Examples

### Начало работы с активными транзакциями

Активные транзакции записи могут применяться к классам модели, а также к экземплярам модели, объекты внутри блока транзакций не обязательно должны быть экземплярами одного класса. Это связано с тем, что транзакции - это подключение по каждой базе данных, а не для каждой модели. Например:

```
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
```

Использование `save` с помощью `bang` гарантирует, что транзакция будет автоматически откатываться при возникновении исключения, и после отката управление переходит к блоку аварийного восстановления для исключения. **Удостоверьтесь, что вы спасаете исключения, выброшенные из сохранения! в блоке транзакций.**

Если вы не хотите использовать `save !`, вы можете вручную поднять `raise ActiveRecord::Rollback` когда сбой не удался. Вам не нужно обрабатывать это исключение. Затем он откатит транзакцию и переведет элемент управления в следующий оператор после блока транзакций.

```
User.transaction do
  if account.save && profile.save
    print "All saves success, returning 1"
    return 1
  else
    raise ActiveRecord::Rollback
  end
end
```

```
end
end
print "Transaction Rolled Back"
```

Прочитайте Активные транзакции ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9326/активные-транзакции-activerecord>

---

# глава 24: Ассоциации ActiveRecord

## Examples

### принадлежит

А ассоциация `belongs_to` устанавливает соединение «один к одному» с другой моделью, поэтому каждый экземпляр объявляющей модели «принадлежит» одному экземпляру другой модели.

Например, если ваше приложение включает пользователей и сообщения, и каждый пост может быть назначен только одному пользователю, вы должны объявить модель сообщения следующим образом:

```
class Post < ApplicationRecord
  belongs_to :user
end
```

В вашей структуре таблицы вы можете

```
create_table "posts", force: :cascade do |t|
  t.integer "user_id", limit: 4
end
```

### has\_one

`has_one` устанавливает соединение «один к одному» с другой моделью, но с другой семантикой. Эта ассоциация указывает, что каждый экземпляр модели содержит или обладает одним экземпляром другой модели.

Например, если каждый пользователь в вашем приложении имеет только одну учетную запись, вы должны объявить модель пользователя следующим образом:

```
class User < ApplicationRecord
  has_one :account
end
```

В ActiveRecord, когда у вас есть отношение `has_one`, активная запись гарантирует, что существует только одна запись с внешним ключом.

Здесь в нашем примере: в таблице учетных записей может быть только одна запись с определенным `user_id`. Если вы пытаетесь связать еще одну учетную запись для одного и того же пользователя, это делает внешний ключ предыдущей записи как `null` (делая ее сиротой) и автоматически создает новую. Это делает предыдущую запись нулевой, даже если сохранение не позволяет новой записи поддерживать согласованность.

```
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]
```

## ИМЕЕТ МНОГО

Связь `has_many` указывает на соединение «один ко многим» с другой моделью. Эта ассоциация, как правило, расположена на другой стороне ассоциации `belongs_to`.

Эта ассоциация указывает, что каждый экземпляр модели имеет ноль или более экземпляров другой модели.

Например, в приложении, содержащем пользователя и сообщения, модель пользователя может быть объявлена следующим образом:

```
class User < ApplicationRecord
  has_many :posts
end
```

Структура таблицы `Post` будет оставаться такой же, как в примере `belongs_to`; в отличие от этого, `User` не будет требовать каких-либо изменений схемы.

Если вы хотите получить список всех опубликованных сообщений для `User`, то вы можете добавить следующее (например, вы можете добавлять области к объектам ассоциации):

```
class User < ApplicationRecord
  has_many :published_posts, -> { where("posts.published IS TRUE") }, class_name: "Post"
end
```

## Полиморфная ассоциация

Этот тип ассоциации позволяет модели `ActiveRecord` принадлежать более чем одному типу записи модели. Общий пример:

```
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
```

Без этой ассоциации у вас будут все эти внешние ключи в вашей таблице адресов, но у вас

будет только значение для одного из них, потому что адрес в этом сценарии может принадлежать только одному объекту (человеку или компании). Вот как это будет выглядеть:

```
class Address < ActiveRecord::Base
  belongs_to :human
  belongs_to :company
end
```

## Has\_many: через ассоциацию

A `has_many :through` объединение часто используется для установления соединения «many-to-many» с другой моделью. Эта ассоциация указывает, что модель объявления может быть сопоставлена с нулем или более экземплярами другой модели, пройдя через третью модель.

Например, рассмотрите медицинскую практику, когда пациенты назначают свидание врачам. Соответствующие декларации ассоциаций могут выглядеть так:

```
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
```

## Has\_one: через ассоциацию

A `has_one :through` ассоциацию устанавливает соединение «one-to-one» с другой моделью. Эта ассоциация указывает, что модель объявления может быть сопоставлена с одним экземпляром другой модели, пройдя через третью модель.

Например, если у каждого `supplier` есть одна `account`, и каждая учетная запись связана с одной учетной записью, то модель поставщика может выглядеть так:

```
class Supplier < ApplicationRecord
  has_one :account
  has_one :account_history, through: :account
end

class Account < ApplicationRecord
  belongs_to :supplier
  has_one :account_history
end
```



```
end

class AccountHistory < ApplicationRecord
  belongs_to :account
end
```

## Ассоциация `has_and_belongs_to_many`

`has_and_belongs_to_many` создает прямое соединение « many-to-many с другой моделью, без промежуточной модели.

Например, если ваше приложение содержит `assemblies` и `parts`, каждая сборка имеет много частей и каждая часть, отображаемая во многих сборках, вы можете объявить модели таким образом:

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

## Ассоциация самореференций

Самореференциальная ассоциация используется для сопоставления модели с самим собой. Наиболее частым примером могло бы быть управление ассоциацией между другом и его последователем.

ex.

```
rails g model friendship user_id:references friend_id:integer
```

теперь вы можете ассоциировать такие модели;

```
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
```

и другая модель будет выглядеть;

```
class Friendship < ActiveRecord::Base
  belongs_to :user
  belongs_to :friend, :class_name => "User"
end
```

Прочитайте Ассоциации ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/1820/>



---

# глава 25: Аутентификация API с помощью Devise

## Вступление

Devise - это решение для проверки подлинности для Rails. Прежде чем идти дальше, я хотел бы добавить краткое описание API. Таким образом, API не обрабатывает сеансы (является апатридом), что означает тот, который обеспечивает ответ после запроса, а затем не требует дополнительного внимания, а это означает, что для работы системы не требуется никакого предыдущего или будущего состояния, всякий раз, когда мы запрашиваем сервер передавать данные аутентификации всем API и сообщать разработчикам, что они не хранят данные аутентификации.

## Examples

### Начиная

Итак, сначала мы создадим проект и устройство настройки рельсов

создать приложение для рельсов

```
rails new devise_example
```

теперь добавьте проект в список драгоценных камней

вы можете найти файл с именем «Gemfile» в корне проекта rails

Затем запустите `bundle install`

Затем вам нужно запустить генератор:

```
rails generate devise:install
```

Теперь на консоли вы можете найти несколько инструкций, просто следуйте за ним.

Создать модель разработки

```
rails generate devise MODEL
```

Затем выполните `rake db:migrate`

Для получения дополнительной информации перейдите по [ссылке](#) : [Devise Gem](#)

# Идентификатор аутентификации

Маркер аутентификации используется для аутентификации пользователя с уникальным токеном. Итак, прежде чем приступить к логике, нам нужно добавить поле `auth_token` в модель `Devise`

Следовательно,

```
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
```

Затем выполните `rake db:migrate`

Теперь мы все настроены на аутентификацию с помощью `auth_token`

В `app/controllers/application_controllers.rb`

Сначала эта строка к ней

```
respond_to :html, :json
```

это поможет приложению rails ответить как html, так и json

затем

```
protect_from_forgery with: :null
```

изменит это `:null` поскольку мы не имеем дело с сеансами.

теперь мы добавим метод проверки подлинности в `application_controller`

Таким образом, по умолчанию `Devise` использует электронную почту как уникальное поле, мы также можем использовать настраиваемые поля, для этого случая мы будем аутентифицироваться с использованием `user_email` и `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
```

Примечание: выше код основан только на вашей логике, я просто пытаюсь объяснить рабочий пример

В строке 6 в приведенном выше коде вы можете увидеть, что я установил `store: false` что предотвратит создание сеанса для каждого запроса, поэтому мы достигли состояния без сохранения

Прочитайте [Аутентификация Апи с помощью Devise онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/9787/аутентификация-апи-с-помощью-devise](https://riptutorial.com/ru/ruby-on-rails/topic/9787/аутентификация-апи-с-помощью-devise)

---

# глава 26: Аутентификация пользователей в Rails

## Вступление

Devise - очень мощный камень, он позволяет вам регистрироваться, входить и выходить из системы сразу после установки. Кроме того, пользователь может добавлять аутентификации и ограничения для своих приложений. У разработчика также есть свои собственные взгляды, если пользователь хочет использовать. Пользователь также может настраивать регистрацию и подписывать формы в соответствии с ее потребностями и требованиями. Следует отметить, что Devise рекомендует вам реализовать свой собственный логин, если вы новичок в рельсах.

## замечания

Во время генерации конфигураций конфигурации с использованием `rails generate devise:install`, `devise` будет перечислять кучу инструкций на терминале для последующего.

Если у вас уже есть модель `USER`, запуск этой команды `rails generate devise USER` добавит необходимые столбцы в существующую модель `USER`.

Используйте этот вспомогательный метод `before_action :authenticate_user!` в верхней части вашего контроллера, чтобы проверить, вошел ли `user` в систему или нет. если нет, то они будут перенаправлены на страницу входа.

## Examples

### Аутентификация с помощью утилиты

Добавить драгоценный камень в Gemfile:

```
gem 'devise'
```

Затем запустите команду `bundle install`.

Используйте команду `$ rails generate devise:install` чтобы сгенерировать требуемый файл конфигурации.

Настройте параметры URL по умолчанию для почтовой программы Devise в каждой среде. В среде разработки добавьте эту строку:

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

к вашему `config/environments/development.rb`

аналогично в этом файле редактирования `config/environments/production.rb` и добавьте

```
config.action_mailer.default_url_options = { host: 'your-site-url' }
```

Затем создайте модель, используя: `$ rails generate devise USER` Где `USER` - это имя класса, для которого вы хотите реализовать аутентификацию.

Наконец, запустите: `rake db:migrate` и вы все настроены.

## Пользовательские виды

Если вам нужно настроить свои представления, вы можете использовать генератор `rails generate devise:views`, который скопирует все представления в ваше приложение. Затем вы можете отредактировать их по своему желанию.

Если в вашем приложении имеется более одной модели Devise (например, `User` and `Admin`), вы заметите, что Devise использует те же представления для всех моделей. Devise предлагает простой способ настройки представлений. Установите `config.scoped_views = true` внутри файла `config/initializers/devise.rb`.

Вы также можете использовать генератор для создания видимых видов: `rails generate devise:views users`

Если вы хотите создать только несколько наборов представлений, например, для регистрируемого и подтверждаемого модуля, используйте флаг `-v`: `rails generate devise:views -v registrations confirmations`

## Конфигурирование фильтров и помощников

Чтобы настроить контроллер с аутентификацией пользователя с помощью `devise`, добавьте это `before_action`: (если ваша модель разработки - «Пользователь»):

```
before_action :authenticate_user!
```

Чтобы проверить, был ли пользователь подписан, используйте следующий помощник:

```
user_signed_in?
```

Для текущего пользователя с подпиской используйте этот помощник:

```
current_user
```

Вы можете получить доступ к сеансу для этой области:

```
user_session
```

- Обратите внимание, что если ваша модель Devise называется `Member` вместо `User`,

замените `user` **ВЫШЕ** на `member`

## OmniAuth

Сначала выберите свою стратегию аутентификации и добавьте ее в свой `Gemfile`. Здесь вы можете найти список стратегий: <https://github.com/intridea/omniauth/wiki/List-of-Strategies>

```
gem 'omniauth-github', :github => 'intridea/omniauth-github'  
gem 'omniauth-openid', :github => 'intridea/omniauth-openid'
```

Вы можете добавить это к своему промежуточному программному обеспечению rails следующим образом:

```
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
```

По умолчанию OmniAuth добавит `/auth/:provider` к вашим маршрутам, и вы можете начать с использования этих путей.

По умолчанию, если есть сбой, omniauth перенаправляет `/auth/failure`

## has\_secure\_password

### Создать модель пользователя

```
rails generate model User email:string password_digest:string
```

### Добавить модуль has\_secure\_password в модель пользователя

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

Теперь вы можете создать нового пользователя с паролем

```
user = User.new email: 'bob@bob.com', password: 'Password1', password_confirmation:  
'Password1'
```

Проверка пароля с помощью метода проверки подлинности

```
user.authenticate('somepassword')
```

## has\_secure\_token



## Создать модель пользователя

```
# Schema: User(token:string, auth_token:string)
class User < ActiveRecord::Base
  has_secure_token
  has_secure_token :auth_token
end
```

Теперь, когда вы создаете нового пользователя, токены и `auth_token` автоматически генерируются

```
user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
user.auth_token # => "77TMHrHJFvFDwodq8w7Ev2m7"
```

Вы можете обновлять маркеры с помощью функции `regenerate_token` и `regenerate_auth_token`

```
user.regenerate_token # => true
user.regenerate_auth_token # => true
```

Прочитайте [Аутентификация пользователей в Rails онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/1794/аутентификация-пользователей-в-rails](https://riptutorial.com/ru/ruby-on-rails/topic/1794/аутентификация-пользователей-в-rails)

# глава 27: Безопасная константа

## Examples

### Успешный `safe_constantize`

User - класс ActiveRecord или Mongoid . Замените User любым классом Rails в вашем проекте (даже что-то вроде Integer или 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]
```

### Неудачный `safe_constantize`

Этот пример не будет работать, потому что строка, переданная внутри, не признается константой в проекте. Даже если вы перейдете в "array" , он не будет работать, поскольку он не заглавный.

```
my_string = "not_a_constant"
# => 'not_a_constant'
my_string.safe_constantize
# => nil

my_string = "array" #Not capitalized!
# => 'array'
my_string.safe_constantize
# => nil
```

Прочитайте Безопасная константа онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3015/безопасная-константа>

---

# глава 28: Безопасное сохранение ключей аутентификации

## Вступление

Многим сторонним API требуется ключ, позволяющий им предотвращать злоупотребления. Если они выдают вам ключ, очень важно, чтобы вы не передавали ключ в общий репозиторий, так как это позволит другим украсть ваш ключ.

## Examples

### Хранение ключей аутентификации с помощью Figaro

Добавьте `gem 'figaro'` в свой Gemfile и запустите `bundle install`. Затем запустите `bundle exec figaro install`; это создаст `config / application.yml` и добавит его в ваш файл `.gitignore`, не допуская его добавления в контроль версий.

Вы можете хранить свои ключи в `application.yml` в таком формате:

```
SECRET_NAME: secret_value
```

где `SECRET_NAME` и `secret_value` - это имя и значение вашего ключа API.

Вы также должны назвать эти секреты в `config / secrets.yml`. У вас могут быть разные секреты в каждой среде. Файл должен выглядеть так:

```
development:
  secret_name: <%= ENV["SECRET_NAME"] %>
test:
  secret_name: <%= ENV["SECRET_NAME"] %>
production:
  secret_name: <%= ENV["SECRET_NAME"] %>
```

Как вы используете эти ключи, зависит, но скажем, например, `some_component` в среде разработки нуждается в доступе к `secret_name`. В `config / environment / development.rb` вы должны поставить:

```
Rails.application.configure do
  config.some_component.configuration_hash = {
    :secret => Rails.application.secrets.secret_name
  }
end
```

Наконец, допустим, вы хотите развернуть производственную среду на Heroku. Эта команда

будет загружать значения в config / environment / production.rb в Heroku:

```
$ figaro heroku:set -e production
```

Прочитайте [Безопасное сохранение ключей аутентификации онлайн](#):

<https://riptutorial.com/ru/ruby-on-rails/topic/9711/безопасное-сохранение-ключей-аутентификации>

---

# глава 29: Блокировка ActiveRecord

## Examples

### Оптимистическая блокировка

```
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
```

### Пессимистическая блокировка

```
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
```

Прочитайте Блокировка ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3866/блокировка-activerecord>

# глава 30: Вложенная форма в Ruby on Rails

## Examples

### Как установить вложенную форму в Ruby on Rails

Первое, что нужно иметь: модель, которая содержит отношение `has_many` к другой модели.

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

**В** `ProjectsController` :

```
class ProjectsController < ApplicationController
  def new
    @project = Project.new
  end
end
```

Во вложенной форме вы можете одновременно создавать дочерние объекты с родительским объектом.

```
<%= 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 %>
```

Как мы инициализируемся `@project` с `Project.new` иметь что - то для создания нового `Project` объекта, точно так же для создания `Todo` объекта, мы должны иметь что - то вроде этого, и есть несколько способов сделать это:

1. В `Projectscontroller` , в `new` методе, вы можете написать: `@todo = @project.todos.build` или `@todo = @project.todos.new` для создания экземпляра нового `Todo` объекта.
2. Вы также можете сделать это в поле зрения: `<%= f.fields_for :todos, @project.todos.build %>`

Для сильных параметров вы можете включить их следующим образом:

```
def project_params
  params.require(:project).permit(:name, todo_attributes: [:name])
end
```

Поскольку объекты `Todo` будут созданы с помощью создания объекта `Project`, вы должны указать эту вещь в модели `Project`, добавив следующую строку:

```
accepts_nested_attributes_for :todos
```

Прочитайте Вложенная форма в Ruby on Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/8203/вложенная-форма-в-ruby-on-rails>

# глава 31: Декоратор

## замечания

**Шаблон Decorator** позволяет добавлять или изменять поведение объектов ситуационным способом, не затрагивая базовый объект.

Это может быть достигнуто, хотя простой Ruby с использованием `stdlib`, или через популярные драгоценные камни, такие как [Draper](#).

## Examples

### Оформление модели с помощью SimpleDelegator

Большинство разработчиков Rails начинают с изменения своей информации о модели в самом шаблоне:

```
<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>
```

Для моделей с большим количеством данных это может быстро стать громоздким и привести к логике копирования вставки из одного шаблона в другой.

В этом примере используется `SimpleDelegator` из `stdlib`.

Все запросы к объекту `SimpleDelegator` по умолчанию передаются родительскому объекту. Вы можете переопределить любой метод с помощью логики представления или добавить новые методы, специфичные для этого представления.

`SimpleDelegator` предоставляет два метода: `__setobj__` для установки того, для какого объекта делегируется, и `__getobj__` для получения этого объекта.

```
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

  # 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
end

```

Некоторые декораторы полагаются на магию, чтобы связать это поведение, но вы можете сделать его более очевидным, откуда приходит логика представления, инициализируя объект на странице.

```

<% user = UserDecorator.new(@user, self) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>

```

Передавая ссылку на объект вида в декоратор, мы все равно можем получить доступ ко всем остальным помощникам вида при построении логики представления без необходимости включать его.

Теперь шаблон просмотра касается только вставки данных на страницу, и это гораздо более понятно.

## Украшение модели с помощью Draper

Дрейпер автоматически согласовывает модели с их декораторами.

```

# 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
end

```

Учитывая переменную `@user` содержащую объект ActiveRecord, вы можете получить доступ к своему декоратору, вызвав `#decorate` в `@user`, или указав класс Draper, если хотите быть конкретным.

```

<% user = @user.decorate %><!-- OR -->
<% user = UserDecorator.decorate(@user) %>
<h1><%= user.full_name %></h1>
<h3>joined: <%= user.created_at %></h3>

```

Прочитайте Декоратор онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/5694/декоратор>

---

# глава 32: Добавить панель администратора

## Вступление

Если вы хотите добавить панель администратора в свое приложение rails, это всего лишь вопрос минут.

## Синтаксис

1. Откройте gem-файл и создайте gem 'rails\_admin', '~> 1.0'
2. установить пакет
3. rails g rails\_admin: установить
4. он спросит вас о маршруте admin, если вы хотите пойти по умолчанию, нажмите Enter.
5. Теперь перейдите в приложение / config / initializers / rails\_admin.rb и вставьте этот код: `config.authorize_with do redirect_to main_app.root_path, если current_user.try (:admin?) End` Этот код позволит только пользователю администратора получить доступ к вашему сайту `yoursite.com/admin` other пользователи будут перенаправлены на корневой путь.
6. Для получения дополнительной информации проверьте документацию этого драгоценного камня. [https://github.com/sferik/rails\\_admin/wiki](https://github.com/sferik/rails_admin/wiki)

## замечания

Используйте его, если вы хотите иметь Admin на свой сайт, иначе нет необходимости в этом. Это проще и мощнее, чем `active_admin gem`. Вы можете добавить это на любой этап после создания пользователей и не забудьте сделать любой пользовательский администратор до 4-го шага. Используйте `sancaп` для предоставления ролей.

## Examples

Итак, вот несколько снимков экрана с панели администратора с использованием `rails_admin gem`.

Как вы можете видеть, макет этого драгоценного камня очень увлекателен и удобен для пользователя.

NAVIGATION



[Blogs](#)

[Users](#)

# Site Administration

Dashboard

 Dashboard

Model name	Last created	Records
<a href="#">Blogs</a>	about 7 hours ago	
<a href="#">Users</a>	about 23 hours ago	

NAVIGATION

Blogs

Users

# List of Users

Dashboard / Users

List

+ Add new

Export

Filter

Refresh



<input type="checkbox"/>	Id	Email	Reset password sent at	Remember c
<input type="checkbox"/>	2	2@gmail.com	-	-
<input type="checkbox"/>	1	1@gmail.com	-	-

2 users

## NAVIGATION

Blogs

Users

# List of Blogs

[Dashboard](#) / [Blogs](#)

☰ List

+ Add new

📄 Export

Filter

↻ Refresh

✕

<input type="checkbox"/>	Id	Title	Content	Created at
<input type="checkbox"/>	7	Post 3	Test content	December 07, 2016 08:19
<input type="checkbox"/>	6	Post 2	test content	December 06, 2016 16:16
<input type="checkbox"/>	5	Post 1	test content	December 06, 2016 16:16

3 blogs

Прочитайте [Добавить панель администратора онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/8128/добавить-панель-администратора](https://riptutorial.com/ru/ruby-on-rails/topic/8128/добавить-панель-администратора)

---

# глава 33: Добавление RDS Amazon к вашему рельсу

## Вступление

Шаги для создания экземпляра AWS RDS и настройки вашего файла `database.yml` путем установки необходимых соединителей.

## Examples

Подумайте, мы подключаем MySQL RDS к вашему рельсовому приложению.

### Шаги по созданию базы данных MySQL

1. Войдите в аккаунт amazon и выберите услугу RDS
2. Выберите « Launch DB Instance на вкладке экземпляра
3. По умолчанию будет выбрано MySQL Community Edition, поэтому нажмите кнопку `select`
4. Выберите цель базы данных, скажем, `production` и нажмите `next step`
5. Предоставьте `mysql version`, `storage size`, `DB Instance Identifier`, `Master Username` and `Password` и нажмите `next step`
6. Введите `Database Name` и нажмите « Launch DB Instance
7. Подождите, пока все экземпляры не будут созданы. После создания экземпляра вы найдете `Endpoint`, скопируйте эту точку входа (которая называется именем хоста)

### Установка разъемов

Добавьте адаптер базы данных MySQL в `gemfile` вашего проекта,

```
gem 'mysql2'
```

Установите свои добавленные драгоценные камни,

```
bundle install
```

Некоторые другие адаптеры баз данных,

- `gem 'pg'` для PostgreSQL
- `gem 'activerecord-oracle_enhanced-adapter'` для Oracle
- `gem 'sql_server'` для SQL Server

**Настройте файл database.yml вашего проекта** Откройте файл config / database.yml

```
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 endpoint
  port: <%= RDS_PORT %> # db port. For MYSQL 3306
```

Прочитайте [Добавление RDS Amazon к вашему рельсу онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/10922/добавление-rds-amazon-к-вашему-рельсу): <https://riptutorial.com/ru/ruby-on-rails/topic/10922/добавление-rds-amazon-к-вашему-рельсу>

---

# глава 34: Драгоценные камни

## замечания

### Документация Gemfile

Для проектов, которые, как ожидается, будут расти, неплохо добавить комментарии к вашему `Gemfile`. Таким образом, даже в больших установках вы все равно знаете, что делает каждый камень, даже если имя не является самоочевидным, и вы добавили его 2 года назад.

Это также поможет вам вспомнить, почему вы выбрали определенную версию и, следовательно, позже оцените ее версию.

Примеры:

```
# temporary downgrade for TeamCity
gem 'rake', '~> 10.5.0'
# To upload invoicing information to payment provider
gem 'net-sftp'
```

### Examples

#### Что такое драгоценный камень?

Драгоценный камень является эквивалентом плагина или расширения для рубина языка программирования.

Точнее, даже рельсы - не что иное, как драгоценный камень. Много драгоценных камней построено на рельсах или других драгоценных камнях (они зависят от этого драгоценного камня) или являются отдельными.

---

## В проекте Rails

### Gemfile

Для вашего проекта Rails у вас есть файл `Gemfile`. Здесь вы можете добавить драгоценные камни, которые вы хотите включить и использовать в своем проекте. После добавления вам необходимо установить драгоценный камень, используя `bundler` (см. Раздел Bundler).

### Gemfile.lock



Как только вы это `Gemfile.lock` , ваш `Gemfile.lock` будет обновлен вашими недавно добавленными драгоценными камнями и их зависимостями. Этот файл блокирует используемые вами драгоценные камни, поэтому они используют эту конкретную версию, объявленную в этом файле.

```
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)
```

Этот пример для гема `devise` . В `Gemfile.lock` версия `4.0.3` , чтобы сообщать при установке вашего проекта на другой машине или на вашем производственном сервере, для которой указанная версия используется.

---

## развитие

Либо один человек, группа или целое сообщество работает и поддерживает драгоценный камень. Выполненная работа, как правило , освобожден после того, как некоторые `issues` были исправлены или `features` , которые были добавлены.

Обычно выпуски следуют принципу [Semantic Versioning 2.0.0](#) .

## Bundler

Самый простой способ обработки и управления драгоценными камнями - использование `bundler` . [Bundler](#) - это менеджер пакетов, сопоставимый с `bower`.

Чтобы использовать пакет, вам сначала нужно его установить.

```
gem install bundler
```

После того как у вас `Gemfile` и запущен `Gemfile` вы должны добавить `gems` в свой `Gemfile` и запустить

```
bundle
```

в вашем терминале. Это устанавливает ваши недавно добавленные драгоценные камни в ваш проект. В случае возникновения проблемы вы получите приглашение в своем терминале.

Если вас интересует более подробная информация, я предлагаю вам ознакомиться с

[документами](#) .

## Gemfiles

Для начала, gemfiles требуют, по крайней мере, один источник, в виде URL-адреса для сервера RubyGems.

Создайте Gemfile с исходным кодом [rubygems.org](https://rubygems.org), запустив `bundle init` . Используйте `https`, поэтому ваше соединение с сервером будет проверено с помощью SSL.

```
source 'https://rubygems.org'
```

Затем объявите драгоценные камни, которые вам нужны, включая номера версий.

```
gem 'rails', '4.2.6'  
gem 'rack', '>=1.1'  
gem 'puma', '~>3.0'
```

Большинство спецификаторов версии, например `> = 1.0`, не требуют пояснений. Спецификатор `~>` имеет особое значение. `~> 2.0.3` идентичен `> = 2.0.3` и `<2.1`. `~> 2.1` совпадает с `> = 2.1` и `<3.0`. `~> 2.2.beta` будет соответствовать предварительным версиям типа `2.2.b.12`.

Репозитории Git также являются достоверными источниками драгоценных камней, если репо содержит один или несколько действующих драгоценных камней. Укажите, что проверить `:tag` `:branch` или `:ref` . По умолчанию используется `master` ветвь.

```
gem 'nokogiri', :git => 'https://github.com/sparklemotion/nokogiri', :branch => 'master'
```

Если вы хотите использовать распакованный камень непосредственно из файловой системы, просто установите путь: путь к пути, содержащему файлы gem.

```
gem 'extracted_library', :path => './vendor/extracted_library'
```

Зависимости могут быть помещены в группы. Группы могут быть проигнорированы во время установки (с использованием `- без --without` ) или требуются все сразу (с использованием `Bundler.require` ).

```
gem 'rails_12factor', group: :production  
  
group :development, :test do  
  gem 'byebug'  
  gem 'web-console', '~> 2.0'  
  gem 'spring'  
  gem 'dotenv-rails'  
end
```

Вы можете указать требуемую версию Ruby в Gemfile с `ruby` . Если Gemfile загружается в

другую версию Ruby, Bundler будет вызывать исключение с объяснением.

```
ruby '2.3.1'
```

## Gemsets

Если вы используете RVM (Ruby Version Manager) то использовать `gemset` для каждого проекта - хорошая идея. `gemset` - это просто контейнер, который вы можете использовать, чтобы держать драгоценные камни отдельно друг от друга. Создание `gemset` каждого проекта позволяет вам изменять драгоценные камни (и версии `gem`) для одного проекта, не нарушая всех ваших других проектов. Каждый проект должен только беспокоиться о своих драгоценных камнях.

RVM обеспечивает ( $> = 0.1.8$ ) `@global gemset` для рубинового интерпретатора. Драгоценные камни, которые вы устанавливаете в `@global gemset` для данного рубина, доступны для всех других гемсет, созданных вами в сочетании с этим рубином. Это хороший способ разрешить всем вашим проектам использовать один и тот же установленный жемчуг для конкретной установки интерпретатора Ruby.

### Создание гемсет

Предположим, что у вас уже установлен `ruby-2.3.1` и вы выбрали его с помощью этой команды:

```
rvm use ruby-2.3.1
```

Теперь, чтобы создать `gemset` для этой рубиновой версии:

```
rvm gemset create new_gemset
```

где `new_gemset` - это имя `gemset`. Чтобы просмотреть список доступных `gemset` для рубиновой версии:

```
rvm gemset list
```

для перечисления драгоценных камней всех рубиновых версий:

```
rvm gemset list_all
```

использовать `gemset` из списка (предположим, что `new_gemset` - это `gemset`, который я хочу использовать):

```
rvm gemset use new_gemset
```

вы также можете указать версию `ruby` с помощью `gemset`, если вы хотите перейти на

другую версию ruby:

```
rvm use ruby-2.1.1@new_gemset
```

для указания gemset по умолчанию для конкретной версии Ruby:

```
rvm use 2.1.1@new_gemset --default
```

чтобы удалить все установленные драгоценные камни из gemset, вы можете его удалить:

```
rvm gemset empty new_gemset
```

чтобы скопировать гемсет из одного рубина в другой, вы можете сделать это:

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

для удаления gemset:

```
rvm gemset delete new_gemset
```

для просмотра текущего имени gemset:

```
rvm gemset name
```

установить жемчужину в глобальном gemset:

```
rvm @global do gem install ...
```

## Инициализация Gemsets во время установки Ruby

Когда вы устанавливаете новый рубин, RVM не только создает два gemset (по умолчанию, пустой gemset и глобальный gemset), он также использует набор редактируемых пользователем файлов, чтобы определить, какие камни для установки.

Работа в `~/.rvm/gemsets`, `rvm`-поиск для `global.gems` и `default.gems` с использованием дерева-иерархии на основе установленной рубиновой строки. Используя пример `ree-1.8.7-p2010.02`, `rvm` проверит (и импортирует из) следующие файлы:

```
~/.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
```

Например, если вы отредактировали `~/.rvm/gemsets/global.gems`, добавив эти две строки:

```
bundler  
awesome_print
```

каждый раз, когда вы устанавливаете новый рубин, эти два драгоценных камня устанавливаются в ваш глобальный гемсет. `default.gems` и файлы `global.gems` обычно перезаписываются во время обновления `rvm`.

Прочитайте Драгоценные камни онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3130/драгоценные-камни>

# глава 35: Загрузка файлов

## Examples

### Загрузка одного файла с использованием Carrierwave

Начать загрузку файлов в Rails довольно просто, прежде всего вам нужно сделать выбор плагина для управления загрузками. Наиболее распространенными являются **Carrierwave** и **Paperclip**. Оба они похожи по функциональности и богаты документацией по

Давайте посмотрим на пример с простым загружаемым изображением аватара с Carrierwave

После `bundle install Carrierwave` введите консоль

```
$ rails generate uploader ProfileUploader
```

Это создаст файл конфигурации, расположенный в `/app/uploaders/profile_uploader.rb`

Здесь вы можете настроить хранилище (то есть локальное или облачное), применять расширения для манипуляций с изображениями (т. Е. Генерировать большие пальцы через MiniMagick) и устанавливать белый список на стороне сервера

Затем создайте новую миграцию с типом строки для `user_pic` и установите загрузчик для него в модели `user.rb`.

```
mount_uploader :user_pic, ProfileUploader
```

Затем отобразите форму для загрузки аватара (это может быть вид редактирования для пользователя)

```
<% 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 %>
```

Обязательно включите `{multipart: true}` в форме заказа, чтобы обрабатывать закладки. Принять необязательно, чтобы установить белый список списка на стороне клиента.

Чтобы отобразить аватар, просто выполните

```
<%= image_tag @user.user_pic.url %>
```

### Вложенная модель - несколько загрузок

Если вы хотите создать несколько загрузок, прежде всего вы можете создать новую модель и установить отношения

Предположим, вы хотите получить несколько изображений для модели продукта. Создайте новую модель и сделайте ее `belongs_to` вашей родительской модели

```
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` необходимо, поскольку он позволяет нам создавать вложенную форму, поэтому мы можем загружать новый файл, изменять имя продукта и устанавливать цену из одной формы

Затем создайте форму в представлении (отредактируйте / создайте)

```
<%= 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 %>
```

Контроллер ничего особенного, если вы не хотите создавать новый, просто создайте новый в своем контроллере продукта

```
# create an action
def upload_file
  printer = Product.find_by_id(params[:id])
  @product_photo = printer.product_photos.create(photo_params)
end

# strong params
private
def photo_params
  params.require(:product_photos).permit(:image)
end
```

Показать все изображения в представлении

```
<% @product.product_photos.each do |i| %>
  <%= image_tag i.image.url, class: 'img-rounded' %>
```

```
<% end %>
```

Прочитайте Загрузка файлов онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2831/загрузка-файлов>



---

# глава 36: Зарезервированные слова

## Вступление

Вы должны быть осторожны, используя эти слова для переменной, имени модели, имени метода или т. Д.

## Examples

### Зарезервированный список слов

- ADDITIONAL\_LOAD\_PATHS
- ФГДД
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- массив
- BasicSocket
- эталонный тест
- Bignum
- переплет
- CGI
- CGIMethods
- cross\_compiling
- Учебный класс
- ClassInheritableAttributes
- сравнимый
- ConditionVariable
- конфиг
- продолжение
- DRB
- DRbIdConv
- DRbObject
- DRbUndumped
- Данные
- Дата
- DateTime
- Delegater
- доверитель
- дайджест
- Dir

- ENV
- EOFError
- Еврорадио
- перечислимый
- Errno
- исключение
- ЛОЖНЫЙ
- FalseClass
- Fcntl
- файл
- FileList
- FileTask
- FileTest
- FileUtils
- Fixnum
- терка
- FloatDomainError
- GC
- драгоценный камень
- GetoptLong
- гашиш
- IO
- IOError
- IPSocket
- IPsocket
- IndexError
- инфлектор
- целое число
- Прерывание
- ядро
- LN\_SUPPORTED
- LoadError
- LocalJumpError
- лесоруб
- маршал
- MatchData
- MatchingData
- математический
- метод
- модуль
- Mutex
- Mysql
- MysqlError
- MysqlField
- MysqlRes

- NIL
- NameError
- NilClass
- NoMemoryError
- NoMethodError
- NoWrite
- NotImplementedError
- числовой
- OPT\_TABLE
- объект
- пространства объектов
- наблюдаемый
- наблюдатель
- PError
- PGconn
- PGLarge
- PGresult
- ПЛАТФОРМА
- PStore
- ParseDate
- точность
- процедура
- Процесс
- Очередь
- RAKEVERSION
- ДАТА ВЫХОДА
- РУБИН
- RUBY\_PLATFORM
- RUBY\_RELEASE\_DATE
- RUBY\_VERSION
- стеллаж
- Грабли
- RakeApp
- RakeFileUtils
- Спектр
- RangeError
- рациональный
- Regexp
- RegexpError
- Запрос
- Ошибка выполнения
- STDERR
- STDIN
- STDOUT
- ScanError

- Ошибка скрипта
- SecurityError
- Сигнал
- SignalException
- SimpleDelegater
- SimpleDelegator
- одиночка
- SizedQueue
- Разъем
- Ошибка сокета
- Стандартная ошибка
- строка
- StringScanner
- Struct
- Условное обозначение
- Ошибка синтаксиса
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TcpSocket
- TCPServer
- TcpSocket
- TOPLEVEL\_BINDING
- ПРАВДА
- задача
- Текст
- Нить
- ThreadError
- ThreadGroup
- Время
- Сделка
- TrueClass
- TypeError
- UdpSocket
- UdpSocket
- UNIXServer
- UNIXSocket
- UNIXserver
- UNIXsocket
- UnboundMethod
- Веб-сайт
- ВЕРСИЯ
- Подробный
- YAML

- ZeroDivisionError
- @base\_path
- принимать
- Доступ
- Акси
- действие
- атрибуты
- Application2
- Позвоните
- категория
- соединение
- база данных
- диспетчер
- DISPLAY1
- привод
- ошибки
- формат
- хозяин
- ключ
- расположение
- нагрузка
- ссылка на сайт
- новый
- поставить в известность
- открыть
- обществу
- котировка
- оказывать
- запрос
- учет
- ответы
- спасти
- объем
- Отправить
- сессия
- система
- шаблон
- тестовое задание
- Тайм-аут
- to\_s
- тип
- URI
- посещения

- наблюдатель

## Имена полей базы данных

- создан в
- создано на
- updated\_at
- обновление
- deleted\_at
- (паранойя
- драгоценный камень)
- lock\_version
- тип
- Я бы
- # {Имя\_таблицы} \_count
- позиция
- parent\_id
- LFT
- полк
- quote\_value

## Рубиновые зарезервированные слова

- псевдоним
- а также
- НАЧАТЬ
- начать
- перерыв
- дело
- учебный класс
- Защита
- определены?
- делать
- еще
- ELSIF
- КОНЕЦ
- конец
- обеспечивать
- ложный
- за
- если
- модуль
- следующий
- ноль

- не
- или же
- переделывать
- спасение
- повторить попытку
- вернуть
- сам
- супер
- затем
- правда
- UNDEF
- если
- до тех пор
- когда
- в то время как
- Уступать
- `_ FILE _`
- `_ LINE _`

Прочитайте Зарезервированные слова онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/10818/зарезервированные-слова>

---

# глава 37: Идентификатор

## Вступление

FriendlyId - это «бульдозер швейцарской армии» для пробок и плагинов постоянной ссылки для Active Record. Он позволяет создавать красивые URL-адреса и работать с удобными для пользователя строками, как если бы они были числовыми идентификаторами. С помощью FriendlyId вы можете легко использовать URL-адреса вашего приложения:

<http://example.com/states/washington>

## Examples

### 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
```

---

## изменить приложение / 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')
```

Новая функция «кандидатов», которая упрощает настройку списка альтернативных слизней, которые могут использоваться для уникального распознавания записей, а не для добавления последовательности. Например:

```
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]
    ]
  end
end
```

Установите длину лимита пули, используя драгоценный камень `friendly_id`?

```
def normalize_friendly_id(string)
  super[0..40]
end
```

Прочитайте Идентификатор онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9664/идентификатор>

---

# глава 38: Изменение стандартного приложения Rails

## Вступление

Это обсудит, как изменить среду, поэтому, когда кто-то создает `rails s` они загружают, а не в среде, которую они хотят.

## Examples

### Работа на локальной машине

Обычно, когда среда рельсов запускается путем ввода. Это просто запускает среду по умолчанию, которая обычно `development`

```
rails s
```

Конкретную среду можно выбрать, используя флаг `-e` например:

```
rails s -e test
```

Что будет запускать тестовую среду.

Окружение по умолчанию можно изменить в терминале, отредактировав файл `~/.bashrc` и добавив следующую строку:

```
export RAILS_ENV=production in your
```

### Работа на сервере

Если на удаленном сервере, использующем Passenger, измените `apache.conf` на среду, которую вы хотите использовать. Например, в этом случае вы видите `RailsEnv production`.

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## This is the default
</VirtualHost>
```

Прочитайте Изменение стандартного приложения Rails онлайн:

<https://riptutorial.com/ru/ruby-on-rails/topic/9915/изменение-стандартного-приложения-rails>

---

# глава 39: Изменение часового пояса по умолчанию

## замечания

`config.active_record.default_timezone` определяет, следует ли использовать `Time.local` (если установлено значение: `local`) или `Time.utc` (если установлено: `utc`) при выводе дат и времени из базы данных. По умолчанию: `utc`.

<http://guides.rubyonrails.org/configuring.html>

---

Если вы хотите изменить часовой пояс **Rails**, но продолжайте сохранять **Active Record** в базе данных в **UTC**, используйте

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

---

Если вы хотите изменить часовой пояс **Rails** и иметь время хранения **Active Record** в этом часовом поясе, используйте

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

**Предупреждение** : вам нужно подумать дважды, даже трижды, перед тем, как сохранять время в базе данных в формате, отличном от UTC.

### Заметка

Не забудьте перезапустить сервер Rails после изменения `application.rb`.

---

Помните, что `config.active_record.default_timezone` может принимать только два значения

- : **local** (преобразовывается в часовой пояс, определенный в `config.time_zone`)
- : **utc** (конвертирует в UTC)

---

Вот как вы можете найти все доступные часовые пояса

```
rake time:zones:all
```

## Examples

## Изменить часовой пояс Rails, но продолжайте сохранять Active Record в базе данных в UTC

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

## Изменение часового пояса Rails и сохранение времени хранения активной записи в этом часовом поясе

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

Прочитайте [Изменение часового пояса по умолчанию онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/3367/изменение-часового-пояса-по-умолчанию): <https://riptutorial.com/ru/ruby-on-rails/topic/3367/изменение-часового-пояса-по-умолчанию>

---

# глава 40: Импортировать все CSV-файлы из определенной папки

## Вступление

В этом примере, скажем, у нас есть много файлов CSV продукта в папке. Каждый CSV-файл должен загрузить нашу базу данных с нашей консоли, напишите команду. Для создания этой модели выполните следующую команду в новом или существующем проекте.

## Examples

### Загрузка CSV из команды консоли

Команды терминала:

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Поздно создайте контроллер.

Команды терминала:

```
rails g controller Products
```

Код контроллера:

```
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
```

Модель:

```
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
```

приложение / Config / application.rb

```
require 'csv'
```

Теперь откройте свою `console` разработки и `run`

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

Прочитайте [Импортировать все CSV-файлы из определенной папки онлайн:](https://riptutorial.com/ru/ruby-on-rails/topic/8658/импортировать-все-csv-файлы-из-определенной-папки)  
<https://riptutorial.com/ru/ruby-on-rails/topic/8658/импортировать-все-csv-файлы-из-определенной-папки>

---

# глава 41: Инструменты для оптимизации и очистки кода Ruby on Rails

## Вступление

Сохранение вашего кода в чистоте и организации при разработке большого приложения Rails может быть довольно сложной задачей даже для опытного разработчика. К счастью, есть целая категория драгоценных камней, которые делают эту работу намного проще.

## Examples

Если вы хотите сохранить ваш код в обслуживании, безопасном и оптимизированном, посмотрите на некоторые драгоценные камни для оптимизации и очистки кода:

### пуля

Это особенно взорвало мой разум. Драгоценный камень пули помогает вам убить все  $N + 1$  запросов, а также излишне нетерпеливые загруженные отношения. После того, как вы установите его и начнете посещать различные маршруты в разработке, появятся предупреждающие поля с предупреждениями, указывающими запросы базы данных, которые необходимо оптимизировать. Он работает прямо из коробки и чрезвычайно полезен для оптимизации вашего приложения.

### Рекомендации по Rails

Анализатор статического кода для обнаружения запахов, специфичных для кода Rails. Он предлагает множество предложений; использовать доступ к области видимости, ограничивать автогенерируемые маршруты, добавлять индексы базы данных и т. д. Тем не менее, в нем содержится много хороших предложений, которые позволят вам лучше взглянуть на то, как перефакторировать свой код и изучить некоторые передовые методы.

### Rubocop

Анализатор статического кода Ruby, который вы можете использовать, чтобы проверить, соответствует ли ваш код правилам сообщества сообщества Ruby. Драгоценные камни сообщают о нарушениях стиля через командную строку с большим количеством полезных свойств рефакторинга кода, таких как бесполезное назначение переменных, избыточное использование `Object # to_s` в интерполяции или даже неиспользуемый аргумент метода.

Хорошо, что он очень настраиваемый, поскольку анализатор может быть весьма раздражающим, если вы не следуете руководству стиля Ruby на 100% (т. Е. У вас много

отстающих пробелов или вы дважды указываете свои строки, даже если не интерполировать и т. Д.), ,

Он разделен на 4 субанализатора (называемых полицейскими): стиль, линт, метрики и рельсы.

Прочитайте [Инструменты для оптимизации и очистки кода Ruby on Rails онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/8713/инструменты-для-оптимизации-и-очистки-кода-ruby-on-rails):

<https://riptutorial.com/ru/ruby-on-rails/topic/8713/инструменты-для-оптимизации-и-очистки-кода-ruby-on-rails>



---

# глава 42: Интеграция React.js с рельсами с использованием Hyperloop

## Вступление

В этом разделе рассматривается интеграция React.js с Rails с использованием [Hyperloop gem](#)

Другие подходы, не охватываемые здесь, - использование ременных рельсов или камней реакции\_он\_ра.

## замечания

Классы компонентов просто генерируют эквивалентные классы компонентов javascript.

Вы также можете получить доступ к компонентам и библиотекам javascript непосредственно из ваших классов компонентов ruby.

Hyperloop будет «prerender» на стороне сервера просмотра, поэтому ваше начальное представление будет загружаться так же, как шаблоны ERB или HAML. После загрузки на клиент реакции берет на себя и будет постепенно обновлять DOM по мере изменения состояния из-за входов от пользователя, HTTP-запросов или входящих данных веб-сокета.

Помимо компонентов, Hyperloop имеет магазины для управления общим состоянием, операции для инкапсуляции изоморфной бизнес-логики и модели, которые дают прямой доступ к вашим моделям ActiveRecord на клиенте с использованием стандартного синтаксиса AR.

Подробнее здесь: <http://ruby-hyperloop.io/>

## Examples

### Добавление простого реагирующего компонента (написанного в рубине) в ваше приложение Rails

1. Добавьте гиперлоп-жемчужину в ваши рельсы (4.0 - 5.1) Gemfile
2. `bundle install`
3. Добавьте манифест гиперлопа в файл `application.js`:

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

#### 4. Создайте свои реагирующие компоненты и поместите их в `hyperloop/components`

```
# 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. Компоненты действуют так же, как и представления. Они «монтируются» с использованием метода `render_component` в контроллере:

```
# somewhere in a controller:
...
def hello_world
  render_component # renders HelloWorld based on method name
end
```

### Объявление параметров компонента (реквизита)

```
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')

# params can be given a default value:
param guest: 'friend' # or
param :guest, default: 'friend'
```

### HTML-теги

```
# HTML tags are built in and are UPPERCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end
```

### Обработчики событий

```
# 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

```

## СОСТОЯНИЯ

```

# 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

```

Обратите внимание, что состояния могут делиться между компонентами с помощью [Hyperloop :: Stores](#)

## 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

```

Прочитайте [Интеграция React.js с рельсами с использованием Hyperloop онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9809/интеграция-react-js-с-рельсами-с-использованием-hyperloop>](https://riptutorial.com/ru/ruby-on-rails/topic/9809/интеграция-react-js-с-рельсами-с-использованием-hyperloop)

---

# глава 43: Интерфейс запросов ActiveRecord

## Вступление

ActiveRecord - это M в MVC, который является уровнем системы, отвечающей за представление бизнес-данных и логики. Метод, который соединяет богатые объекты приложения с таблицами в системе управления реляционными базами данных, - это **O bject R elational M apper ( ORM )**.

ActiveRecord будет выполнять запросы в базе данных для вас и совместим с большинством систем баз данных. Независимо от того, какую систему баз данных вы используете, формат метода ActiveRecord всегда будет таким же.

## Examples

### .где

Метод `where` доступен для любой модели ActiveRecord и позволяет запрашивать базу данных для набора записей, соответствующих заданным критериям.

Метод `where` принимает хэш, где ключи соответствуют именам столбцов в таблице, представленной моделью.

В качестве простого примера мы будем использовать следующую модель:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

Чтобы найти всех людей с первым именем `Sven` :

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

Чтобы найти всех людей с именем `Sven` и фамилией `Schrodinger` :

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

В приведенном выше примере вывод `sql` показывает, что записи будут возвращаться только в том случае, если совпадают имя `first_name` И `last_name` .

## запрос с условием ИЛИ

Чтобы найти записи с `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')
```

## . где с массивом

Метод `where` для любой модели ActiveRecord может использоваться для генерации SQL формы `WHERE column_name IN (a, b, c, ...)`. Это достигается передачей массива в качестве аргумента.

В качестве простого примера мы будем использовать следующую модель:

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end

people = Person.where(first_name: ['Mark', 'Mary'])
people.to_sql # "SELECT * FROM people WHERE first_name IN ('Mark', 'Mary')"
```

Если массив содержит `nil`, то SQL будет изменено, чтобы проверить, если столбец является `null`:

```
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"
```

## Области применения

Области действия действуют как предопределенные фильтры на моделях ActiveRecord.

Область определения определяется с помощью метода класса `scope`.

В качестве простого примера мы будем использовать следующую модель:

```
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 можно вызывать непосредственно из класса модели:

```
minors = Person.minors
```

Области могут быть связаны:

```
peters_children = Person.minors.with_last_name('Peters')
```

Метод `where` и другие методы типа запроса также могут быть связаны:

```
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

За кулисами области - это просто синтаксический сахар для стандартного метода класса. Например, эти методы функционально идентичны:

```
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

## Область по умолчанию

в вашей модели, чтобы установить область по умолчанию для всех операций над моделью.

Существует одна заметная разница между методом `scope` методом класса: `scope - defined scopes всегда` возвращают `ActiveRecord::Relation`, даже если логика внутри возвращает `nil`. Однако методы класса не имеют такой защитной сетки и могут разрушать цепочки, если они возвращают что-то еще.

## where.not

where clauses можно `where.not` синтаксиса `where.not` :

```
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')
```

Поддерживается ActiveRecord 4.0 и более поздних версий.

## заказ

Вы можете заказать результаты запроса **ActiveRecord** с помощью `.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">]
```

Если не указано, заказы будут выполняться в порядке возрастания. Вы можете указать его, выполнив:

```
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` также принимает строку, поэтому вы также можете сделать

```
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">]
```

Поскольку строка является сырым SQL, вы также можете указать таблицу, а не только атрибут. Предполагая, что вы хотите заказать `users` соответствии с их именем `role`, вы можете сделать это:

```
Class User < ActiveRecord::Base
  belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

Область `order` также может принимать узел Arel:

```
User.includes(:role).order(User.arel_table[:name].asc)
```

## Методы ActiveRecord Bang (!)

Если вам нужен метод **ActiveRecord** для создания исключения вместо `false` значения в случае сбоя, вы можете добавить `!` им. Это очень важно. Поскольку некоторые исключения / неудачи трудно поймать, если вы не используете `!` на них. Я рекомендовал сделать это в вашем цикле разработки, чтобы написать весь ваш код ActiveRecord таким образом, чтобы сэкономить ваше время и проблемы.

```
Class User < ActiveRecord::Base
  validates :last_name, presence: true
end
```

```
User.create!(first_name: "John")
#=> ActiveRecord::RecordInvalid: Validation failed: Last name can't be blank
```

Методами **ActiveRecord**, которые принимают *взрыва* ( ! ), Являются:

- .create!
- .take!
- .first!
- .last!
- .find\_by!
- .find\_or\_create\_by!
- #save!
- #update!
- все динамические искатели AR

## .find\_by

Вы можете найти записи по любому полю в своей таблице, используя `find_by`.

Итак, если у вас есть модель `User` с атрибутом `first_name` вы можете:

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

`find_by` что `find_by` по умолчанию не выбрасывает исключение. Если результатом является пустой набор, он возвращает `nil` вместо `find`.

Если это исключение, вы можете использовать `find_by!` что вызывает ошибку

`ActiveRecord::RecordNotFound` например `find`.

## .удалить все

Если вам нужно быстро удалить много записей, **ActiveRecord** предоставляет метод `.delete_all` для вызова непосредственно на модели, для удаления всех записей в этой таблице или коллекции. Остерегайтесь, хотя, поскольку `.delete_all` не создает экземпляра какого-либо объекта, следовательно, не предоставляет никакого обратного вызова ( `before_*` и `after_destroy` не запускаются).

```
User.delete_all
#=> 39 <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

## Нечувствительный к регистру поиск по регистру ActiveRecord

Если вам нужно искать модель ActiveRecord для аналогичных значений, у вас может возникнуть соблазн использовать `LIKE` или `ILIKE` но это не переносится между `ILIKE` базы



данных. Точно так же, прибегая к постоянному снижению или росту, вы можете создавать проблемы с производительностью.

Вы можете использовать метод `matches` ActiveR для ActiveRecord, чтобы сделать это безопасным способом:

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

Age1 применит соответствующую конструкцию LIKE или ILIKE для сконфигурированного ядра базы данных.

## Получить первую и последнюю запись

У Rails есть очень простой способ получить `first` и `last` запись из базы данных.

Чтобы получить `first` запись из таблицы `users` нам нужно ввести следующую команду:

```
User.first
```

Он будет генерировать следующий запрос `sql` :

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC LIMIT 1
```

И вернется следующая запись:

```
#<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 >
```

Чтобы получить `last` запись из таблицы `users` нам нужно ввести следующую команду:

```
User.last
```

Он будет генерировать следующий запрос `sql` :

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` DESC LIMIT 1
```

И вернется следующая запись:

```
#<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 >
```

Передача целочисленного значения **первому** и **последнему** методу создает запрос **LIMIT** и возвращает массив объектов.

```
User.first(5)
```

Он будет генерировать следующий запрос `sql` .

```
SELECT "users".* FROM "users" ORDER BY "users"."id" ASC LIMIT 5
```

А также

```
User.last(5)
```

Он будет генерировать следующий запрос `sql` .

```
SELECT "users".* FROM "users" ORDER BY "users"."id" DESC LIMIT 5
```

## .группа и .count

У нас есть модель `Product` и мы хотим сгруппировать их по их `category` .

```
Product.select(:category).group(:category)
```

Это запросит базу данных следующим образом:

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

Убедитесь, что поле сгруппировано также выбрано. Группирование особенно полезно для подсчета появления - в данном случае - `categories` .

```
Product.select(:category).group(:category).count
```

Как показывает запрос, он будет использовать базу данных для подсчета, что намного эффективнее, чем сначала получить всю запись, и сделать подсчет в коде:

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS products_category FROM "products" GROUP BY "products"."category"
```

## .distinct (или .uniq)

Если вы хотите удалить дубликаты из результата, вы можете использовать `.distinct()` :

```
Customers.select(:country).distinct
```

Это запрашивает базу данных следующим образом:

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()` имеет тот же эффект. С Rails 5.0 он устарел, и он будет удален из Rails с версией 5.1. Причина в том, что слово `unique` не имеет того же значения, что и отдельный, и оно

может вводить в заблуждение. Кроме того, `distinct` подход ближе к синтаксису SQL.

## присоединяется

`joins()` позволяет присоединить таблицы к текущей модели. Напр.

```
User.joins(:posts)
```

будет вызывать следующий SQL-запрос:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id"
```

После соединения таблицы у вас будет доступ к ней:

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

Обратите внимание на множественную форму. Если ваше отношение `:has_many`, то аргумент `join` `joins()` должен быть плюрализован. В противном случае используйте единственное число.

Вложенные `joins`:

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

который будет производить:

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER JOIN "images" ON "images"."post_id" = "images"."id"
```

## Включает в себя

`ActiveRecord` с `includes` гарантирует, что все указанные ассоциации загружаются с использованием минимально возможного количества запросов. Поэтому при запросе таблицы для данных со связанной таблицей обе таблицы загружаются в память.

```
@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 })` загрузит только **авторов** с условиями в память **без загрузки книг**. Используйте `joins` если дополнительная информация о вложенных ассоциациях не требуется.

```
@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
```

## Ограничение и смещение

Вы можете использовать `limit` чтобы указать количество записей, которые нужно извлечь, и использовать `offset` чтобы сообщить количество пропущенных записей, прежде чем начинать возвращать записи.

Например

```
User.limit(3) #returns first three records
```

Он будет генерировать следующий запрос sql.

```
"SELECT `users`.* FROM `users` LIMIT 3"
```

Поскольку смещение не упоминается в вышеуказанном запросе, оно возвращает первые три записи.

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

Он будет генерировать следующий запрос sql.

```
"SELECT `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

Прочитайте Интерфейс запросов ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2154/интерфейс-запросов-activerecord>

---

# глава 44: Использование GoogleMaps с Rails

## Examples

### Добавьте тег javascript Google Maps в заголовок макета

Чтобы карты Google корректно работали с [turbolinks](#) , добавьте тег javascript непосредственно в заголовок макета, а не включите его в представление.

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

Значение `google_maps_api_script_tag` лучше всего определено в помощнике.

```
# 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
```

Вы можете зарегистрировать свое приложение в Google и получить свой ключ [api в консоли google api](#) . В Google есть краткое [руководство о том, как запросить ключ api для javascript api](#) .

Ключ api хранится в файле `secrets.yml` :

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

Не забудьте добавить `config/secrets.yml` в ваш `.gitignore` файл и убедитесь, что вы не

передаете ключ api в репозиторий.

## Геокодировать модель

Предположим, что у ваших пользователей и / или групп есть профили, и вы хотите отображать поля профиля адреса на карте Google.

```
# app/models/profile_fields/address.rb
class ProfileFields::Address < ProfileFields::Base
  # Attributes:
  # label, e.g. "Work address"
  # value, e.g. "Willy-Brandt-Straße 1\n10557 Berlin"
end
```

Отличный способ геокодирования адресов, то есть обеспечить longitude и latitude - это [драгоценный камень геокодирования](#).

Добавьте геокодер в свой Gemfile и запустите bundle чтобы установить его.

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

Добавьте столбцы базы данных для latitude и longitude, чтобы сохранить местоположение в базе данных. Это более эффективно, чем запрос службы геокодирования каждый раз, когда вам нужно место. Это быстрее, и вы не так быстро достигаете предела запроса.

```
→ 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
```

Добавьте механизм геокодирования к вашей модели. В этом примере строка адреса хранится в атрибуте value. Настройте геокодирование для выполнения, когда запись изменилась, и только то, что имеет значение:

```
# 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
```

По умолчанию геокодер использует google как службу поиска. Он имеет множество интересных функций, таких как дистанционные вычисления или поиск близости. За дополнительной информацией обратитесь к [геокодированию README](#).

## Показывать адреса на карте google в профиле

В представлении профиля отобразите поля профиля пользователя или группы в списке, а также поля адреса на карте google.

```
- # app/views/profiles/show.html.haml
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

Соответствующие `@profile_fields` и `@address_fields` задаются в контроллере:

```
# 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
```

Инициализируйте карту, поместите маркеры, установите масштаб и другие настройки карты с помощью javascript.

Info **Contact** Corporate Vita Work & Study More


## Contact Information

Email [doe@example.com](mailto:doe@example.com)

Work address  
John Doe  
Willy-Brandt-Straße 1  
10557 Berlin

Work address  
John Doe  
1-6 Chesham Pl  
London  
SW1X 8PZ  
United Kingdom

Phone 123 456



## Установите маркеры на карте с помощью javascript

Предположим, что есть `div .google_map`, который станет картой и который имеет поля адреса, которые будут отображаться как маркеры как атрибут `data`.

Например:

```

<!-- 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>

```

Чтобы использовать событие `$(document).ready` с [turbolinks](#) без управления событиями [turbolinks](#) вручную, используйте [jquery.turbolinks gem](#).

Если вы хотите выполнить некоторые другие операции с картой, позже, например, для фильтрации или инфо-окна, удобно иметь карту, управляемую [классом сценария кофе](#).



```
# 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
```

При использовании нескольких файлов сценариев кофе, которые по умолчанию являются именами, удобно определять глобальное пространство имен `App`, которое используется всеми файлами кофейных скриптов.

Затем `.google_map` (возможно, несколько) `.google_map` и создайте один экземпляр класса `App.GoogleMap` для КАЖДОГО ИЗ НИХ.

```
# 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
```

## Инициализируйте карту, используя класс сценария кофе.

При условии [класса сценария кофе-скрипта](#) `App.GoogleMap` карту google можно инициализировать следующим образом:

```
# 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
}
```

Чтобы узнать больше о возможных параметрах `map_configuration`, просмотрите [документацию MapOptions Google](#) и их [руководство по добавлению элементов управления](#).

Для справки, класс `google.maps.Map` широко документирован [здесь](#).

## Инициализация маркеров карты с использованием класса сценария кофе

При условии, что `App.GoogleMap` [сценария кофе](#) `App.GoogleMap` и информация о маркере, хранящаяся в атрибуте `data-address-fields` в `div.google_map`, маркеры карт могут быть инициализированы на карте следующим образом:

```
# 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
```

Чтобы узнать больше о параметрах маркера, [ознакомьтесь с документацией Google MarkerOptions](#) и их [руководством к маркерам](#).

## Автоматическое масштабирование карты с использованием класса

## сценария кофе

При условии, что `App.GoogleMap` сценария кофе-класса `App.GoogleMap` содержит `App.GoogleMap` `google.maps.Map` хранящуюся как `@map` а `@map google.maps.Marker` хранится как `@markers`, карта может быть автоматически увеличена, то есть отрегулирована, чтобы все маркеры были видны, например : на карте:

```
# 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
```

Чтобы узнать больше о границах, посмотрите [документацию Google LatLngBounds](#).

## Отображение свойств модели как json

Чтобы отображать поля профиля адреса в качестве маркеров на карте google, объекты поля адреса должны передаваться как json-объекты в javascript.

---

# Обычные атрибуты базы данных

При вызове `to_json` объекта `ApplicationRecord` автоматически отображаются атрибуты базы данных.

Учитывая модель `ProfileFields::Address` с атрибутами `label`, `value`, `longitude` и `latitude`, `address_field.as_json` приводит к `Hash`, например, представлению,

```
address_field.as_json # =>
{label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
 longitude: ..., latitude: ...}
```

который преобразуется в строку json `to_json`:

```
address_field.to_json # =>
{"label\":"Work address\","value\":"Willy-Brandt-Straße 1\\n
10557 Berlin\","longitude\":"...","latitude\":"..."}
```

Это полезно, потому что позволяет использовать `label` и `value` позже в javascript, например, чтобы показать подсказки инструментов для маркеров карты.

---

## Другие атрибуты

Другие виртуальные атрибуты могут быть раскрыты путем переопределения метода `as_json`

Например, чтобы открыть атрибут `title`, `as_json` его в объединенный хэш `as_json`:

```
# 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
```

В приведенном выше примере `super` используется для вызова исходного метода `as_json`, который возвращает исходный хэш атрибута объекта и объединяет его с требуемым хэш-`as_json` позиции.

Чтобы понять разницу между `as_json` и `to_json`, посмотрите на [это сообщение в блоге jjulian](#)

---

## Позиция

Для рендеринга маркеров google maps api по умолчанию требует хэш `position` который имеет долготу и широту, хранящиеся как `lng` и `lat` соответственно.

Эта позиция hash может быть создана в javascript, позже или здесь при определении json-представления поля адреса:

Чтобы обеспечить эту `position` как json-атрибут поля адреса, просто переопределите метод `as_json` на модели.

```
# 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
```

Прочитайте Использование GoogleMaps с Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2828/использование-googlemaps-с-rails>

# глава 45: Команды генерации Rails

## Вступление

Использование: `rails generate GENERATOR_NAME [args] [options]` .

Используйте `rails generate` для отображения доступных генераторов. Псевдоним: `rails g` .

## параметры

параметр	подробности
<code>-h / --help</code>	Получить помощь по любой команде генератора
<code>-p / --pretend</code>	Режим притвора: запустите генератор, но не создавайте и не изменяйте файлы
<code>field:type</code>	«field-name» - это имя создаваемого столбца, а «type» - тип столбца данных. Возможные значения для типа «type» в <code>field:type</code> приведены в разделе «Примечания».

## замечания

Возможные значения для типа «type» в `field:type` :

Тип данных	Описание
<code>:string</code>	Для меньших фрагментов текста (обычно имеет предел символов 255)
<code>:text</code>	Для более длинных фрагментов текста, как абзац
<code>:binary</code>	Хранение данных, включая изображения, аудио и видео
<code>:boolean</code>	Сохранение истинных или ложных значений
<code>:date</code>	Только дата
<code>:time</code>	Только время
<code>:datetime</code>	Дата и время
<code>:float</code>	Хранение поплавков без точности
<code>:decimal</code>	Хранение поплавков с точностью

Тип данных	Описание
:integer	Сохранение целых чисел

## Examples

### Rails Generate Model

Чтобы создать модель ActiveRecord которая автоматически создает правильные db-миграции и тестовые файлы шаблонов для вашей модели, введите эту команду

```
rails generate model NAME column_name:column_type
```

«NAME» - это имя модели. «field» - это имя столбца в таблице DB, а «type» - это тип столбца (например, `name:string` или `body:text` ). Просмотрите раздел «Примечания» для списка поддерживаемых типов столбцов.

Чтобы настроить внешние ключи, добавьте `belongs_to:model_name` .

Поэтому скажите, что вы хотите настроить модель User которая имеет `username` , `email` и принадлежит к `School` , вы должны ввести следующую

```
rails generate model User username:string email:string school:belongs_to
```

`rails g` является сокращением для `rails generate` . Это приведет к такому же результату

```
rails g model User username:string email:string school:belongs_to
```

### Rails Generate Migration

Вы можете создать файл миграции рельсов с терминала, используя следующую команду:

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

Для списка всех параметров, поддерживаемых этой командой, вы можете запустить команду без каких-либо аргументов, как в `rails generate migration` .

Например, если вы хотите добавить поля `first_name` и `last_name` в таблицу `users` , вы можете сделать следующее:

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Rails создаст следующий файл миграции:

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
```

```
def change
  add_column :users, :last_name, :string
  add_column :users, :first_name, :string
end
end
```

Теперь примените ожидающие миграции к базе данных, запустив в терминале следующее:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

**Примечание.** Для еще меньшего набора текста вы можете заменить `generate` на `g`

## Rails Generate Scaffold

**ОТКАЗ ОТ ОТВЕТСТВЕННОСТИ:** строительные леса не рекомендуется, если только для обычных приложений CRUD / тестирования. Это может генерировать много файлов (представлений / моделей / контроллеров), которые не нужны в вашем веб-приложении, тем самым вызывая головные боли (bad :!).

Чтобы создать полностью работающий эшафот для нового объекта, включая модель, контроллер, представления, активы и тесты, используйте команду `rails g scaffold`.

```
$ 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
  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
```

Затем вы можете запустить `rake db:migrate` чтобы настроить таблицу базы данных.

Затем вы можете посетить <http://localhost:3000/widgets>, и вы увидите полностью функциональный CRUD-эшафот.

## Контроллер контроллера Rails

мы можем создать новый контроллер с командой `rails g controller`.

```
$ bin/rails generate controller controller_name
```

Генератор контроллера ожидает параметры в виде генератора `generate controller ControllerName action1 action2`.

Следующее создает контроллер `Greetings` с действием `hello`.

```
$ bin/rails generate controller Greetings hello
```

Вы увидите следующий вывод:

```
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
```

Это генерирует следующее

файл	пример
Файл контроллера	<code>greetings_controller.rb</code>
Просмотреть файл	<code>hello.html.erb</code>
Функциональный тестовый файл	<code>greetings_controller_test.rb</code>

файл	пример
Просмотреть помощника	<code>greetings_helper.rb</code>
Файл JavaScript	<code>greetings.coffee</code>

Он также добавит маршруты для каждого действия в `routes.rb`

Прочитайте Команды генерации Rails онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2540/команды-генерации-rails>

# глава 46: Консолидация активов

## Вступление

Конвейент ресурсов обеспечивает структуру для конкатенации и минимизации или сжатия ресурсов JavaScript и CSS. Он также добавляет возможность писать эти активы на других языках и предварительных процессорах, таких как CoffeeScript, Sass и ERB. Это позволяет автоматически использовать активы в вашем приложении вместе с активами других драгоценных камней. Например, jquery-rails включает в себя копию jquery.js и включает функции AJAX в Rails.

## Examples

### Рейк-задачи

По умолчанию sprockets-rails поставляется со следующими рейк-задачами:

- `assets:clean[keep]` : удалить старые скомпилированные активы
- `assets:clobber` : удалить скомпилированные активы
- `assets:environment` : загрузить среду компиляции ресурсов
- `assets:precompile` : Скомпилировать все активы, названные в `config.assets.precompile`

### Манифестные файлы и директивы

В `assets initializer` (`config/initializers/assets.rb`) несколько файлов явно определен прекомпилировать.

```
# Precompile additional assets.
# application.coffee, application.scss, and all non-JS/CSS in app/assets folder are already
added.
# Rails.application.config.assets.precompile += %( search.js )
```

В этом примере `application.coffee` и `application.scss` называются так называемыми файлами манифеста. Эти файлы должны использоваться для включения других ресурсов JavaScript или CSS. Доступны следующие команды:

- `require <path>` : `require` директивных функций , аналогичных рубин собственного `require` . Он предоставляет способ объявить зависимость от файла в вашем пути и гарантирует, что он будет загружен только один раз перед исходным файлом.
- `require_directory <path>` : требует наличия всех файлов внутри одного каталога. Он похож на `path/*` поскольку он не соответствует вложенным каталогам.
- `require_tree <path>` : требует наличия всех вложенных файлов в каталоге. Его глобальным эквивалентом является `path/**/*` .

- `require_self` : заставляет тело текущего файла , который будет вставлен перед любой последующий `require` директивы. Полезно в CSS-файлах, где общий индексный файл содержит глобальные стили, которые необходимо определить до загрузки других зависимостей.
- `stub <path>` : удалить файл из включенного
- `depend_on <path>` : позволяет указать зависимость от файла без его включения. Это используется для кеширования. Любые изменения, внесенные в файл зависимостей, аннулируют кеш исходного файла.

Файл `application.scss` может выглядеть так:

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

Другим примером является файл `application.coffee` . Здесь с включением `jquery` и `Turbolinks` :

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

Если вы не используете CoffeeScript, но обычный JavaScript, синтаксис будет выглядеть так:

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

## Основное использование

Существует два основных способа использования конвейера активов:

1. При запуске сервера в режиме разработки он автоматически обрабатывает и подготавливает ваши активы «на лету».
2. В режиме производства вы, вероятно, будете использовать его для предварительной обработки, модификации и сжатия и компиляции ваших активов. Вы можете сделать это, выполнив следующую команду:

```
bundle exec rake assets:precompile
```

Прочитайте [Консолидация активов онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/3386/консолидация-активов](https://riptutorial.com/ru/ruby-on-rails/topic/3386/консолидация-активов)

---

# глава 47: конфигурация

## Examples

### Пользовательская конфигурация

Создайте файл `YAML` каталоге `config/` , например: `config/neo4j.yml`

Содержимое `neo4j.yml` может быть чем-то вроде ниже (для простоты по `default` используется для всех сред):

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

**В** `config/application.rb` :

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

Теперь ваша настраиваемая конфигурация доступна, как показано ниже:

```
Rails.configuration.neo4j['host']
#=> localhost
Rails.configuration.neo4j['port']
#=> 7474
```

---

### Больше информации

Официальный документ API Rails описывает метод `config_for` как:

Удобство для загрузки `config / foo.yml` для текущего Rails env.

---

**Если вы не хотите использовать файл `yaml`**

Вы можете настроить свой собственный код через объект конфигурации Rails с помощью настраиваемой конфигурации в соответствии с свойством `config.x`.

## пример

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.x.super_debugger = true
```

Эти точки конфигурации затем доступны через объект конфигурации:

```
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
```

Прочитайте конфигурация онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2558/>  
конфигурация

# глава 48: конфигурация

## Examples

### Среды в Rails

Файлы конфигурации для рельсов можно найти в `config/environments/`. По умолчанию рельсы имеют 3 среды, `development`, `production` и `test`. Редактируя каждый файл, вы редактируете конфигурацию только для этой среды.

Rails также имеет конфигурационный файл в `config/application.rb`. Это общий файл конфигурации, так как любые параметры, указанные здесь, перезаписываются конфигурацией, указанной в каждой среде.

Вы добавляете или изменяете параметры конфигурации в блоке `Rails.application.configure do` и параметрах конфигурации, начиная с `config`.

### Настройка базы данных

Конфигурация базы данных проекта rails лежит в файле `config/database.yml`. Если вы создаете проект с использованием `rails new` команды `rails new` и не указываете механизм базы данных, который будет использоваться, тогда rails использует `sqlite` в качестве базы данных по умолчанию. Типичный файл `database.yml` с конфигурацией по умолчанию будет похож на следующий.

```
# 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
```

Если вы хотите изменить базу данных по умолчанию при создании нового проекта, вы можете указать базу данных: `rails new hello_world --database=mysql`

## Общая конфигурация Rails

Следующие параметры конфигурации следует вызывать в объекте `Rails::Railtie`

- **config.after\_initialize** : принимает блок, который будет запущен после того, как рельсы инициализируют приложение.
- **config.asset\_host** : Это устанавливает хост для активов. Это полезно при использовании *сети доставки контента* . Это сокращенно для `config.action_controller.asset_host`
- **config.autoload\_once\_paths** : Этот параметр принимает массив путей, в которых Rails автоматически загружает константы. По умолчанию используется пустой массив
- **config.autoload\_paths** : Принимает массив путей, в которых Rails автоматически загружает константы. По умолчанию все каталоги в `app`
- **config.cache\_classes** : определяет, следует ли перезагружать классы и модули по каждому запросу. В режиме разработки это значение по умолчанию равно `false` а в режимах производства и тестирования по умолчанию оно равно `true`
- **config.action\_view.cache\_template\_loading** : Это определяет, следует ли перезагружать шаблоны по каждому запросу. По умолчанию используется настройка `config.cache_classes`
- **config.beginning\_of\_week** : устанавливает начальное значение по умолчанию в начале недели. Для этого требуется действительный символ дня недели ( `:monday` )
- **config.cache\_store** : выберите, какой кеш-память использовать. Параметры включают в себя `:file_store` `:memory_store` , `mem_cache_store` **или** `null_store` .
- **config.colorize\_logging** : это определяет, **расписана** ли информация о протоколировании
- **config.eager\_load** : загружает все зарегистрированные
- **config.encoding** : Указывает кодировку приложения. Значение по умолчанию - `UTF-8`
- **config.log\_level** : Устанавливает многословие регистратора Rails. По умолчанию используется `:debug` во всех средах.
- **config.middleware** : используйте это для настройки промежуточного программного обеспечения приложения
- **config.time\_zone** : устанавливает часовой пояс приложения по умолчанию.

## Настройка активов

Следующие параметры конфигурации могут использоваться для настройки активов

- **config.assets.enabled** : Определяет, включен ли конвейер активов. Это значение по умолчанию равно `true`



- **config.assets.raise\_runtime\_errors** : Это позволяет проверять время выполнения. Это полезно для `development mode`
- **config.assets.compress** : позволяет сжать активы. В режиме производства это значение по умолчанию равно `true`
- **config.assets.js\_compressor** : Указывает, какой JS-компрессор использовать. Варианты включают `:closure` `:uglifyer` И `:yui`
- **config.assets.paths** : Указывает, какие пути для поиска активов.
- **config.assets.precompile** : позволяет выбрать дополнительные активы, которые будут предварительно скомпилированы при использовании `rake assets:precompile` запускается
- **config.assets.digest** : этот параметр позволяет использовать отпечатки пальцев MD-5 в именах активов. Значение по умолчанию равно `true` в режиме разработки
- **config.assets.compile**: Переключает жить `Sprockets` компиляции в рабочем режиме

## Настройка генераторов

Rails позволяет вам настроить, какие генераторы используются при запуске команд `rails generate` . Этот метод, `config.generators` принимает блок

```
config.generators do |g|
  g.orm :active_record
  g.test_framework :test_unit
end
```

Вот некоторые из вариантов

вариант	Описание	По умолчанию
активы	Создает активы при создании леса	правда
force_plural	Позволяет использовать множественные имена моделей	ложный
помощник	Определяет, создавать ли помощники	правда
integration_tool	Укажите инструмент интеграции	test_unit
javascript_engine	Настраивает JS-движок	:js
resource_route	Создает маршрут ресурса	правда
stylesheet_engine	Настраивает движок стилей	:cs
scaffold_stylesheet	Создает CSS для строительных лесов	правда
test_framework	Укажите тестовую структуру	Minitest

вариант	Описание	По умолчанию
template_engine	Настраивает механизм шаблонов	:erb

Прочитайте конфигурация онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2841/конфигурация>

# глава 49: Кроватька PDF

## Examples

### Расширенный пример

Это расширенный подход с примером

```
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
```

Я над кодом у нас есть эта строка `FundsPdf.new(@fund, view_context)`. Здесь мы инициализируем класс `FundsPdf` с экземпляром `@fund` и `view_context` для использования вспомогательных методов в `FundsPdf`. `FundsPdf` wuld выглядят так

```
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
  end
end
```

```
text "Thank you for your order.Print this receipt as
confirmation of your order.",
  :indent_paragraphs => 40, :size => 13
end
end
```

Это один из лучших подходов к созданию PDF-файлов с использованием уроков креветок.

## Основной пример

Вам нужно добавить Gem и PDF MIME: Введите внутри `mime_types.rb`, поскольку нам нужно уведомить рельсы о типе PDF-типа.

После этого мы можем сгенерировать Pdf с креветкой следующими основными способами

---

## Это основное назначение

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

---

## Мы можем сделать это с помощью неявного блока

```
Prawn::Document.generate("implicit.pdf") do
  text "Hello World"
end
```

---

## С явным блоком

```
Prawn::Document.generate("explicit.pdf") do |pdf|
  pdf.text "Hello World"
end
```

Прочитайте Креветка PDF онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/4163/креветка-pdf>

# глава 50: Кэширование

## Examples

### Кэширование русской куклы

Вы можете захотеть вложить в кэшированные фрагменты внутри других кэшированных фрагментов. Это называется `Russian doll caching`.

Преимущество `Russian doll caching` заключается в том, что при обновлении одного продукта все остальные внутренние фрагменты могут быть повторно использованы при регенерации внешнего фрагмента.

Как объяснялось в предыдущем разделе, кэшированный файл истечет, если значение `updated_at` изменится для записи, от которой напрямую зависит кэшированный файл. Однако это не приведет к истечению срока хранения кеша, в который вложен фрагмент.

Например, сделайте следующее представление:

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

Что, в свою очередь, отражает это мнение:

```
<% cache game do %>
  <%= render game %>
<% end %>
```

Если какой-либо атрибут игры будет изменен, значение `updated_at` будет установлено на текущее время, тем самым заканчивая кеш.

Однако, поскольку `updated_at` не будет изменен для объекта продукта, этот кеш не будет истек, и ваше приложение будет обслуживать устаревшие данные. Чтобы исправить это, мы связываем модели вместе с сенсорным методом:

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

### Кэширование SQL

Кэширование запросов - это функция `Rails` которая кэширует набор результатов, возвращаемый каждым запросом. Если `Rails` снова встретит тот же запрос для этого запроса, он будет использовать кэшированный результирующий набор в отличие от повторного запуска запроса к базе данных.

Например:

```
class ProductsController < ApplicationController

  def index
    # Run a find query
    @products = Product.all

    ...

    # Run the same query again
    @products = Product.all
  end

end
```

Во второй раз, когда тот же запрос выполняется с базой данных, он фактически не попадет в базу данных. В первый раз, когда результат возвращается из запроса, он сохраняется в кеше запросов (в памяти), а во второй раз он извлекается из памяти.

Однако важно отметить, что кешы запросов создаются в начале действия и уничтожаются в конце этого действия и, таким образом, сохраняются только на время действия. Если вы хотите сохранить результаты запроса более устойчивым образом, вы можете с низким уровнем кэширования.

## Кэширование фрагментов

`Rails.cache`, предоставляемый `ActiveSupport`, может использоваться для кэширования любого сериализуемого объекта `Ruby` через запросы.

Чтобы извлечь значение из кеша для данного ключа, используйте `cache.read`:

```
Rails.cache.read('city')
# => nil
```

Используйте `cache.write` для записи значения в кеш:

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

В качестве альтернативы, используйте `cache.fetch` для чтения значения из кеша и необязательно напишите по умолчанию, если нет значения:

```
Rails.cache.fetch('user') do
  User.where(:is_awesome => true)
end
```

Возвращаемое значение переданного блока будет присвоено кешу под заданным ключом, а затем возвращено.

Вы также можете указать срок действия кеша:

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

## Кэширование страницы

Вы можете использовать кеш страницы [page\\_caching](#) для кэширования отдельных страниц. Это сохраняет результат одного динамического запроса как статический HTML-файл, который служит вместо динамического запроса при последующих запросах. README содержит полные инструкции по установке. После настройки используйте `cache_page` класса `cache_page` в контроллере для кэширования результата действия:

```
class UsersController < ActionController::Base
  cache_page :index
end
```

Используйте `expire_page` для принудительного истечения срока хранения кеша, удалив сохраненный файл HTML:

```
class UsersController < ActionController::Base
  cache_page :index

  def index
    @users = User.all
  end

  def create
    expire_page :action => :index
  end
end
```

Синтаксис `expire_page` имитирует синтаксис `url_for` и друзей.

## HTTP-кэширование

Rails >= 3 поставляется с возможностями кэширования HTTP из коробки. Это использует заголовки `Cache-Control` и `Etag` чтобы контролировать, как долго клиент или посредник (например, CDN) могут кэшировать страницу.

В действии контроллера используйте `expires_in` для установки длины кэширования для

этого действия:

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

Используйте `expires_now` для принудительного немедленного истечения кэшированного ресурса для любого клиента или посредника:

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```

## Кэширование действий

Подобно кэшированию страниц, кэширование действий кэширует всю страницу. Разница в том, что запрос попадает в стек Rails, поэтому перед запуском фильтров перед тем, как кеш будет обслуживаться. Он извлекается из Rails в [actionpack-action\\_caching gem](#).

Общим примером является кэширование действия, требующего аутентификации:

```
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
```

Параметры включают `:expires_in`, `custom :cache_path` (для действий с несколькими маршрутами, которые должны кэшироваться по-разному) и `:if` / `:unless` контролировать, когда действие должно быть кэшировано.

```
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
```

Когда макет имеет динамический контент, кешируйте только содержимое действия, передавая `layout: false`.

Прочитайте Кэширование онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2833/кэширование>



# глава 51: маршрутизация

## Вступление

Маршрутизатор Rails распознает URL-адреса и отправляет их в действие контроллера. Он также может генерировать пути и URL-адреса, избегая необходимости жестких строк в ваших представлениях.

## замечания

«Маршрутизация» в целом - это то, как URL-адреса «обрабатываются» вашим приложением. В случае Rails обычно это контроллер, и какое действие этого контроллера будет обрабатывать определенный входящий URL. В приложениях Rails маршруты обычно помещаются в файл `config/routes.rb`.

## Examples

### Маршрутизация ресурсов (основная)

Маршруты определяются в `config/routes.rb`. Они часто определяются как группа связанных маршрутов, используя `resources` или методы `resource`.

`resources :users` создают следующие семь маршрутов, все сопоставление с действиями `UserController`:

```
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'
```

Названия действий отображаются после # в `to` параметра выше. Методы с теми же именами должны быть определены в `app/controllers/users_controller.rb` следующим образом:

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

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

Вы можете ограничить действия, которые генерируются `only` или `except` :

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

Вы можете просмотреть все маршруты своего приложения в любое время, запустив:

5.0

```
$ rake routes
```

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
```

Чтобы увидеть только маршруты, которые отображаются на конкретный контроллер:

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
```

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
```

Вы можете выполнять поиск по маршрутам с помощью опции `-g` . Это показывает любой маршрут, который частично соответствует имени вспомогательного метода, URL-адресу или HTTP-глаголу:

5.0

```
$ rake routes -g new_user      # Matches helper method
```

```
$ rake routes -g POST # Matches HTTP Verb POST
```

## 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
```

Кроме того, при запуске сервера `rails` в режиме разработки вы можете получить доступ к веб-странице, которая отображает все ваши маршруты с фильтром поиска, соответствующим приоритету сверху вниз, в `<hostname>/rails/info/routes`. Это будет выглядеть так:

помощник	HTTP-глагол	Дорожка	Контроллер # Действие
Путь / URL		[Матч по пути]	
users_path	ПОЛУЧИТЬ	/users(.:format)	пользователи индекс #
	СООБЩЕНИЕ	/users(.:format)	пользователей # создать
new_user_path	ПОЛУЧИТЬ	/users/new(.:format)	пользователи # новых
edit_user_path	ПОЛУЧИТЬ	/users/:id/edit(.:format)	пользователи # редактировать
user_path	ПОЛУЧИТЬ	/users/:id(.:format)	Пользователи # показать
	PATCH	/users/:id(.:format)	пользователи # обновление
	ПОЛОЖИЛ	/users/:id(.:format)	пользователи # обновление
	УДАЛЯТЬ	/users/:id(.:format)	пользователей # уничтожить

Маршруты могут быть объявлены доступными только для членов (а не коллекций) с использованием `resource` метода вместо `resources` в `routes.rb`. С `resource` маршрут `index` создается не по умолчанию, а только при явном запросе такого типа:

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

## Ограничения

Вы можете фильтровать маршруты, используя ограничения.

Существует несколько способов использования ограничений, в том числе:

- [сегментные ограничения](#) ,

- [ограничения на основе запроса](#)
- [расширенные ограничения](#)

Например, запрошенное ограничение на ограничение позволяет только конкретному IP-адресу разрешить доступ к маршруту:

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

[См. Другие аналогичные примеры ActionController::Routing::Mapper::Scoping](#) .

Если вы хотите сделать что-то более сложное, вы можете использовать более сложные ограничения и создать класс для переноса логики:

```
# 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
```

## Одна форма, несколько кнопок отправки

Вы также можете использовать значение тегов отправки формы как ограничение для перехода к другому действию. Если у вас есть форма с несколькими кнопками отправки (например, «предварительный просмотр» и «отправка»), вы можете зафиксировать это ограничение непосредственно на своих `routes.rb` вместо того, чтобы писать javascript для изменения URL-адреса формы. Например, с камнем [commit\\_param\\_routing](#) вы можете воспользоваться rails `submit_tag`

## Первый параметр Rails `submit_tag` позволяет изменить значение параметра фиксации формы

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

## Маршрутные маршруты

Rails предоставляет несколько способов организации ваших маршрутов.

### Объем по URL-адресу :

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

Это генерирует следующие маршруты

```
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'
```

На стороне сервера может иметь смысл хранить некоторые представления в другой подпапке, чтобы отделять просмотры администраторов от пользовательских представлений.

### Объем по модулю

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

module ищет файлы контроллера под подпапкой данного имени

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

Вы можете переименовать префикс помощника пути, добавив параметр `as`

```
scope 'admin', as: :administration do
  get 'dashboard'
end

# => administration_dashboard_path
```

Rails обеспечивает удобный способ сделать все вышеописанное, используя метод `namespace`. Следующие объявления эквивалентны

```
namespace :admin do
  end

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

## Область применения контроллера

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

Это генерирует эти маршруты

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

## Неглубокое гнездование

Маршруты ресурсов допускают `:shallow` параметр, который помогает, по возможности, сократить URL-адреса. Ресурсы не должны быть вложены более чем на один уровень. Один из способов избежать этого - создать мелкие маршруты. Цель состоит в том, чтобы оставить сегменты сегментов родительской коллекции, где они не нужны. Конечным результатом является то, что только генерируемые вложенные маршруты предназначены для `:index` `:create` и `:new` действий. Остальные сохраняются в собственном неглубоком контексте URL. Существует два варианта возможностей для настраиваемых неглубоких маршрутов:

- **`:shallow_path`: Префикс** пути элемента с заданным параметром

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

- : **shallow\_prefix** : добавьте указанные параметры в именованные помощники

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

Мы также можем проиллюстрировать `shallow` маршруты:

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

альтернативно кодируется следующим образом (если вы счастливы с блоком):

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

Результирующие маршруты:

Префикс	глагол	Шаблон URI
bid_comments	ПОЛУЧИТЬ	/bids/:bid_id/comments(.:format)
	СООБЩЕНИЕ	/bids/:bid_id/comments(.:format)
new_bid_comment	ПОЛУЧИТЬ	/bids/:bid_id/comments/new(.:format)
edit_comment	ПОЛУЧИТЬ	/comments/:id/edit(.:format)
комментарий	ПОЛУЧИТЬ	/comments/:id(.:format)
	ПАТЧН	/comments/:id(.:format)
	ПОЛОЖИЛ	/comments/:id(.:format)
	УДАЛЯТЬ	/comments/:id(.:format)

Префикс	глагол	Шаблон URI
auction_bids	ПОЛУЧИТЬ	/auctions/:auction_id/bids(.:format)
	СООБЩЕНИЕ	/auctions/:auction_id/bids(.:format)
new_auction_bid	ПОЛУЧИТЬ	/auctions/:auction_id/bids/new(.:format)
edit_bid	ПОЛУЧИТЬ	/bids/:id/edit(.:format)
предложение	ПОЛУЧИТЬ	/bids/:id(.:format)
	PATCH	/bids/:id(.:format)
	ПОЛОЖИЛ	/bids/:id(.:format)
	УДАЛЯТЬ	/bids/:id(.:format)
аукционы	ПОЛУЧИТЬ	/auctions(.:format)
	СООБЩЕНИЕ	/auctions(.:format)
new_auction	ПОЛУЧИТЬ	/auctions/new(.:format)
edit_auction	ПОЛУЧИТЬ	/auctions/:id/edit(.:format)
торг	ПОЛУЧИТЬ	/auctions/:id(.:format)
	PATCH	/auctions/:id(.:format)
	ПОЛОЖИЛ	/auctions/:id(.:format)
	УДАЛЯТЬ	/auctions/:id(.:format)

Если вы тщательно проанализируете маршруты, вы заметите, что вложенные части URL-адреса включены только тогда, когда они необходимы для определения того, какие данные будут отображаться.

## Обеспокоенность

Чтобы избежать повторения на вложенных маршрутах, проблемы обеспечивают отличный способ совместного использования общих ресурсов, которые можно повторно использовать. Чтобы создать беспокойство использовать метод `concern` внутри `routes.rb` файла. Метод ожидает символ и блок:

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



Не создавая никаких маршрутов, этот код позволяет использовать атрибут `:concerns` ресурса. Самый простой пример:

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

Эквивалентный вложенный ресурс будет выглядеть так:

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

Это создало бы, например, следующие маршруты:

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

Чтобы проблемы были значимыми, необходимо наличие нескольких ресурсов, которые используют эту проблему. Дополнительные ресурсы могут использовать любой из следующих синтаксисов для вызова проблемы:

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

## Перенаправление

Вы можете перенаправлять маршруты Rails следующим образом:

4,0

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

4,0

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

Вы также можете перенаправить все неизвестные маршруты на заданный путь:

4,0

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

4,0

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

## Маршруты участников и коллекций

Определение блока-члена внутри ресурса создает маршрут, который может действовать на отдельном члене этого маршрута на основе ресурсов:

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

Это генерирует следующий маршрут участника:

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

Маршруты сбора позволяют создавать маршруты, которые могут действовать в коллекции объектов ресурсов:

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

Это создает следующий маршрут сбора:

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

Альтернативный синтаксис:

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

## URL-параметры с периодом

Если вы хотите поддерживать параметр url более сложным, чем номер id, вы можете столкнуться с проблемой с парсером, если значение содержит период. Все, что следует за периодом, будет считаться форматом (т.е. json, xml).

Вы можете обойти это ограничение, используя ограничение для *расширения* принятого ввода.

Например, если вы хотите ссылаться на запись пользователя по адресу электронной почты в URL-адресе:

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

## Корневой маршрут

Вы можете добавить маршрут домашней страницы в свое приложение с помощью `root` метода.

```
# 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
```

А в терминале `rake routes` (`rake routes rails routes` в Rails 5) будут производить:

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

Поскольку главная страница, как правило, является самым важным маршрутом, а маршруты приоритетны в том порядке, в котором они появляются, `root` маршрут обычно должен быть первым в вашем файле маршрутов.

## Дополнительные действия RESTful

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

Это создает следующие маршруты в дополнение к маршрутам RESTful по умолчанию 7 :

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

Если вы хотите сделать это для отдельных строк, вы можете использовать:

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

Вы также можете добавить действие к `/new` пути:

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

Что создаст:

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

Будьте внимательны при добавлении действий на маршруты RESTful, возможно, вам не хватает другого ресурса!

## Область доступных областей

Если ваше приложение доступно на разных языках, вы обычно показываете текущую локаль в URL-адресе.

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

Ваш корень будет доступен через локали, определенные в `I18n.available_locales`.

## Смонтировать другое приложение

`mount` используется для монтирования другого приложения (в основном, стойки) или рельсовых двигателей, которые будут использоваться в текущем приложении

**синтаксис:**

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

Теперь вы можете получить доступ к установленному выше приложению с помощью вспомогательного маршрута `some_rack_app_path` или `some_rack_app_url`.

Но если вы хотите переименовать это имя помощника, вы можете сделать это как:

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

Это создаст `myapp_path` и `myapp_url` помощники, которые можно использовать для перехода к этому подключенному приложению.

## Перенаправления и подстановочные маршруты

Если вы хотите предоставить URL-адрес из удобства для своего пользователя, но

сопоставьте его напрямую с другим, который вы уже используете. Используйте перенаправление:

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

Ну, это стало интересно быстро. Основной принцип здесь заключается в том, чтобы просто использовать метод `#redirect` для отправки одного маршрута на другой маршрут. Если ваш маршрут довольно прост, это действительно простой метод. Но если вы хотите также отправить исходные параметры, вам нужно сделать немного гимнастики, взяв параметр внутри `{here}`. Обратите внимание на одиночные кавычки вокруг всего.

В приведенном выше примере мы также переименовали маршрут для удобства, используя псевдоним с параметром: `as`. Это позволяет использовать это имя в методах, таких как помощники `#_path`. Опять же, проверьте свои `$ rake routes` с вопросами.

## Разделить маршруты на несколько файлов

Если ваш файл маршрутов чрезвычайно большой, вы можете поместить свои маршруты в несколько файлов и включить каждый из файлов с помощью метода `require_relative` Ruby:

`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
```

## Вложенные маршруты

Если вы хотите добавить вложенные маршруты, вы можете написать следующий код в `routes.rb`.

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

Это создаст следующие маршруты:

```
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
```

Прочитайте маршрутизация онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/307/маршрутизация>

# глава 52: Миграции ActiveRecord

## параметры

Тип столбца	Описание
:primary_key	Основной ключ
:string	Более короткий тип данных строки. Позволяет установить <code>limit</code> для максимального количества символов.
:text	Более длинное количество текста. Разрешает <code>limit</code> для максимального количества байтов.
:integer	Integer. Разрешает <code>limit</code> для максимального количества байтов.
:bigint	Большее целое число
:float	терка
:decimal	Десятичное число с переменной точностью. Позволяет настроить <code>precision</code> и <code>scale</code> .
:numeric	Позволяет настроить <code>precision</code> и <code>scale</code> .
:datetime	Объект DateTime для дат / времени.
:time	Объект времени для времени.
:date	Объект даты для дат.
:binary	Двоичные данные. Разрешает <code>limit</code> для максимального количества байтов.
:boolean	логический

## замечания

- Большинство файлов миграции находятся в каталоге `db/migrate/`. Они идентифицируются по метке времени UTC в начале имени файла:  
`YYYYMMDDHHMMSS_create_products.rb`.
- Команда `rails generate` может быть сокращена до `rails g`.

- Если а `:type` не передается в поле, по умолчанию используется строка.

## Examples

### Запуск определенной миграции

Чтобы выполнить определенную миграцию вверх или вниз, используйте `db:migrate:up` или `db:migrate:down`.

Выполнить конкретную миграцию:

5.0

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

5.0

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

Вниз по определенной миграции:

5.0

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

5.0

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

Номер версии в приведенных выше командах - это числовой префикс в имени файла миграции. Например, чтобы перейти на миграцию `20160515085959_add_name_to_users.rb`, вы должны использовать `20160515085959` в качестве номера версии.

### Создать таблицу соединений

Чтобы создать таблицу соединений между `students` и `courses`, выполните команду:

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

Это приведет к следующей миграции:

```
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
```



## Выполнение миграции в разных средах

Чтобы выполнить миграцию в `test` среде, запустите эту команду оболочки:

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

### 5.0

Начиная с Rails 5.0, вы можете использовать `rails` вместо `rake` :

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

## Добавить новый столбец в таблицу

Чтобы добавить новое `name` столбца в таблицу `users` , выполните команду:

```
rails generate migration AddNameToUsers name
```

Это создает следующую миграцию:

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

Когда имя миграции имеет вид `AddXXXXToTABLE_NAME` за которым следует список столбцов с типами данных, сгенерированная миграция будет содержать соответствующие инструкции `add_column` .

## Добавить новый столбец с индексом

Чтобы добавить новое *индексированное* `email` столбца в таблицу `users` , выполните команду:

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

Это приведет к следующей миграции:

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

## Удалить существующий столбец из таблицы

Чтобы удалить существующее `name` столбца из таблицы `users` , выполните команду:

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

Это приведет к следующей миграции:

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

Когда имя миграции имеет вид `RemoveXXXFromYYY` за которым следует список столбцов с типами данных, тогда сгенерированная миграция будет содержать соответствующие инструкции `remove_column` .

Хотя не обязательно указывать тип данных (например `:string` ) в качестве параметра `remove_column` , настоятельно рекомендуется. Если тип данных *не* указан, то миграция не будет обратимой.

## Добавить столбец ссылки в таблицу

Чтобы добавить ссылку на `team` в таблицу `users` , выполните следующую команду:

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

Это создает следующую миграцию:

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

Эта миграция создаст столбец `team_id` в таблице `users` .

Если вы хотите добавить соответствующий `index` и `foreign_key` в добавленный столбец, измените команду на `rails generate migration AddTeamRefToUsers team:references:index` чтобы `rails generate migration AddTeamRefToUsers team:references:index` . Это приведет к следующей миграции:

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

Если вы хотите указать свой ссылочный столбец, отличный от того, что генерирует Rails

автоматически, добавьте следующее в свою миграцию: (Например: вы можете позвонить User который создал Post качестве Author в таблице Post )

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

## Создать новую таблицу

Чтобы создать новую таблицу users с name столбца и salary , выполните команду:

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

Это приведет к следующей миграции:

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

Когда имя миграции имеет вид CreateXXX за которым следует список столбцов с типами данных, тогда будет создана миграция, которая создает таблицу xxx с указанными столбцами.

## Добавление нескольких столбцов в таблицу

Чтобы добавить несколько столбцов в таблицу, разделите field:type пары с пробелами при использовании команды rails generate migration .

Общий синтаксис:

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

Например, следующее добавит поля name , salary и email в таблицу users :

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

Что генерирует следующую миграцию:

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

```
    add_column :users, :email, :string
  end
end
```

## Выполнение миграций

Команда запуска:

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

При указании целевой версии будут выполняться требуемые миграции (вверх, вниз, изменение) до тех пор, пока она не достигнет указанной версии. Здесь `version number` - это числовой префикс в имени файла миграции.

5.0

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

5.0

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

## Откат миграции

`rollback` последней миграции, либо путем возврата метода `change` либо путем использования метода `down`. Команда запуска:

5.0

```
rake db:rollback
```

5.0

```
rails db:rollback
```

## Откат последних трех миграций

5.0

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

5.0

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

STEP предоставляет количество возвращаемых миграций.

## Откат всех миграций

### 5.0

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

### 5.0

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

## Изменение таблиц

Если вы создали таблицу с неправильной схемой, тогда самый простой способ изменить столбцы и их свойства - `change_table` . Просмотрите следующий пример:

```
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
```

Вышеуказанная миграция изменяет `orders` таблиц. Ниже приведены пошаговые описания изменений:

1. `t.remove :ordered_at` удаляет столбец `ordered_at` из `orders` таблицы.
2. `t.string :skew_number` добавляет новый столбец типа `string` с именем `skew_number` в таблице `orders` .
3. `t.index :skew_number` добавляет индекс в столбец `skew_number` в таблице `orders` .
4. `t.rename :location, :state` переименовывает `location` столбца в `orders` таблицы в `state` .

## Добавить уникальный столбец в таблицу

Чтобы добавить новое *уникальное* `email` для `users` , выполните следующую команду:

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

Это создаст следующую миграцию:

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

```
end
```

## Изменить тип существующего столбца

Чтобы изменить существующий столбец в Rails с переносом, выполните следующую команду:

```
rails g migration change_column_in_table
```

Это создаст новый файл миграции в каталоге `db/migration` (если он еще не существует), который будет содержать файл с префиксом `timestamp` и имя файла миграции, которое содержит следующее содержимое:

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

[Rails Guide - Изменение столбцов](#)

## Более длинный, но более безопасный метод

Вышеупомянутый код не позволяет пользователю откатиться от миграции. Вы можете избежать этой проблемы, разделив метод `change` на отдельные методы `up` и `down`:

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

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

## Повторные миграции

Вы можете выполнить откат, а затем снова `redo` миграцию с помощью команды `redo`. Это, в основном, ярлык, сочетающий `rollback` и `migrate` задач.

Команда запуска:

5.0

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

5.0

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

Вы можете использовать параметр `STEP` для возврата более одной версии.

Например, чтобы вернуться 3 миграции:

5.0

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

5.0

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

## Добавить столбец со значением по умолчанию

Следующий пример добавляет `admin` столбца в таблицу `users` и дает этому столбцу значение по умолчанию `false`.

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

Миграции со значениями по умолчанию могут занять много времени в больших таблицах, например, PostgreSQL. Это связано с тем, что каждая строка должна быть обновлена со значением по умолчанию для вновь добавленного столбца. Чтобы обойти это (и сократить время простоя во время развертывания), вы можете разделить свою миграцию на три этапа:

1. Добавьте `add_column` -migration, аналогичный приведенному выше, но не задайте значение по умолчанию
2. Развертывание и обновление столбца в задаче `rake` или на консоли во время работы вашего приложения. Убедитесь, что ваше приложение уже записывает данные в этот `column` для новых / обновленных строк.
3. Добавьте еще одну `change_column`, которая затем изменит значение по умолчанию этого столбца на желаемое значение по умолчанию

## Запретить нулевые значения

Чтобы запретить `null` значения в столбцах таблицы, добавьте параметр `:null` в вашу миграцию, например:

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

## Проверка статуса миграции

Мы можем проверить статус миграции, выполнив

3,0 5,0

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

5.0

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

Результат будет выглядеть так:

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 *****

В поле «Статус» `up` означает, что миграция выполнена и `down` означает, что нам нужно выполнить миграцию.

## Создать столбец `hstore`

`Hstore` могут быть полезны для сохранения настроек. Они доступны в базах данных PostgreSQL после включения расширения.

```
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
```

## Добавить самостоятельную ссылку

Самостоятельная ссылка может быть полезна для построения иерархического дерева. Это может быть достигнуто с помощью `add_reference` в переносе.

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



```
end
end
```

Столбец внешнего ключа будет `pages_id` . Если вы хотите определить имя столбца внешнего ключа, сначала нужно создать столбец и добавить ссылку после.

```
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
```

## Создать столбец массива

Столбец `array` поддерживается PostgreSQL. Rails автоматически преобразует массив PostgreSQL в массив Ruby и наоборот.

Создайте таблицу со столбцом `array` :

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

Добавьте столбец `array` в существующую таблицу:

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

Добавьте индекс для столбца `array` :

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

## Добавление ограничения NOT NULL к существующим данным

Предположим, вы хотите добавить внешний ключ `company_id` в таблицу `users` , и вы хотите ограничить `NOT NULL` . Если у вас уже есть данные у `users` , вам придется сделать это несколькими шагами.

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

```
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
```

Прочитайте Миграции ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/1087/миграции-activerecord>

---

# глава 53: Многоцелевые столбцы ActiveRecord

## Синтаксис

- `serialize: <field_plural_symbol>`

## Examples

### Сохранение объекта

Если у вас есть атрибут, который необходимо сохранить и получить в базе данных в качестве объекта, укажите имя этого атрибута с помощью метода `serialize` и он будет обрабатываться автоматически.

Атрибут должен быть объявлен как `text` поле.

В модели вы должны объявить тип поля ( `Hash` или `Array` )

Дополнительная информация: [serialize >> apidock.com](https://apidock.com)

## Как

---

## В вашей миграции

```
class Users < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ...
      t.text :preference
      t.text :tag
      ...
      t.timestamps
    end
  end
end
```

---

## В вашей модели

```
class User < ActiveRecord::Base
  serialize :preferences, Hash
  serialize :tags, Array
end
```

Прочитайте Многоцелевые столбцы ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/7602/многоцелевые-столбцы-activerecord>

# глава 54: Модельные состояния: AASM

## Examples

### Основное состояние с AASM

Обычно вы создадите модели, которые будут содержать состояние, и это состояние будет меняться в течение срока службы объекта.

[AASM](#) - это библиотека конечных состояний, которая может помочь вам справиться с легким прохождением процесса проектирования ваших объектов.

Наличие чего-то подобного в вашей модели вполне согласуется с идеей [Fat Model](#), [Skinny Controller](#), одной из лучших практик Rails. Модель является единственным ответственным за управление своим состоянием, его изменениями и генерированием событий, вызванных этими изменениями.

Для установки в Gemfile

```
gem 'aasm'
```

Рассмотрим приложение, в котором пользователь заказывает продукт по цене.

```
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
end
```

```
end

private

def set_payment_date
  update payed_at: Time.zone.now
end

end
```

Считается, что состояния класса `Quote` могут идти, но это лучше всего для вашего процесса.

Вы можете думать о состояниях как о прошлом, как в предыдущем примере или в другом в другом времени, например: ценообразование, оплата, доставка и т. Д. Именование государств зависит от вас. С личной точки зрения, прошлые состояния работают лучше, потому что ваше конечное состояние, безусловно, будет прошлым действием и лучше свяжется с именами событий, которые будут объяснены позже.

**ПРИМЕЧАНИЕ.** Будьте осторожны, какие имена вы используете, вам нужно беспокоиться о том, что нельзя использовать зарезервированные ключевые слова Ruby или Ruby on Rails, такие как `valid`, `end`, `being` и т. Д.

Определив состояния и переходы, мы можем теперь получить доступ к некоторым методам, созданным AASM.

Например:

```
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.
```

Поскольку вы видите, что события имеют переходы, эти переходы определяют способ изменения состояния при вызове события. Если событие недействительно из-за текущего состояния, возникает ошибка.

События и переходы также имеют некоторые другие обратные вызовы, например

```
guard: product_delivered?
```

`product_delivered?` метод, который возвращает логическое значение. Если это окажется ложным, переход не будет применяться, и если другие переходы не будут доступны, состояние не изменится.

```
success: :reverse_charges
```

Если этот перевод успешно выполняется, будет вызван метод `:reverse_charges`.

Существует несколько других методов в AASM с большим количеством обратных вызовов в процессе, но это поможет вам создать свои первые модели с конечными состояниями.

Прочитайте **Модельные состояния: AASM онлайн**: <https://riptutorial.com/ru/ruby-on-rails/topic/7826/модельные-состояния--aasm>

# глава 55: Наследование отдельных таблиц

## Вступление

Наследование отдельных таблиц (STI) - это шаблон проектирования, основанный на идее сохранения данных нескольких моделей, которые наследуются от одной и той же базовой модели, в одну таблицу в базе данных.

## Examples

### Основной пример

Сначала нам нужна таблица для хранения наших данных

```
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
```

Затем давайте создадим некоторые модели

```
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
```

Когда вы выполните `Guest.create(name: 'Bob')` ActiveRecord переведет это, чтобы создать запись в таблице Users с `type: 'Guest'`.

Когда вы извлекаете запись `bob = User.where(name: 'Bob').first` возвращаемый объект будет



экземпляром `Guest` , который может быть принудительно обработан как Пользователь с `bob.becomes(User)`

становится наиболее полезным при работе с разделенными частицами или маршрутами / контроллерами суперкласса, а не подкласса.

## Пользовательский столбец наследования

По умолчанию имя класса модели STI хранится в столбце с именем `type` . Но его имя можно изменить, переопределив значение `inheritance_column` в базовом классе. Например:

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

Миграция в этом случае будет выглядеть следующим образом:

```
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
```

Когда вы `Admin.create` , эта запись будет сохранена в таблице `users` с `entity_type = "Admin"`

## Модель Rails с столбцом типа и без ИППП

Наличие столбца `type` в модели Rails без вызова STI может быть достигнуто путем назначения `:_type_disabled` для `inheritance_column` :

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

Прочитайте [Наследование отдельных таблиц онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/9125/наследование-отдельных-таблиц](https://riptutorial.com/ru/ruby-on-rails/topic/9125/наследование-отдельных-таблиц)

---

## глава 56: Настройка углового с рельсами

### Examples

Угловая с рельсами 101

---

## Шаг 1. Создайте новое приложение Rails.

```
gem install rails -v 4.1
rails new angular_example
```

---

## Шаг 2: Удалите Turbolinks

Удаление turbolinks требует удаления из Gemfile.

```
gem 'turbolinks'
```

Удалите `require` из `app/assets/javascripts/application.js` :

```
//= require turbolinks
```

---

## Шаг 3: добавьте AngularJS в конвейер АКТИВОВ

Чтобы заставить Angular работать с конвейером Rails, нам нужно добавить в Gemfile:

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

Теперь запустите команду

```
bundle install
```

Добавьте `bower` чтобы мы могли установить зависимость AngularJS:

```
rails g bower_rails:initialize json
```

Добавить Angular to `bower.json` :

```
{
  "name": "bower-rails generated dependencies",

  "dependencies": {

    "angular": "latest",
    "angular-resource": "latest",
    "bourbon": "latest",
    "angular-bootstrap": "latest",
    "angular-ui-router": "latest"
  }
}
```

Теперь, когда `bower.json` настроен с правильными зависимостями, давайте их установим:

```
bundle exec rake bower:install
```

---

## Шаг 4: Упорядочение углового приложения

Создайте следующую структуру папок в `app/assets/javascript/angular-app/` :

```
templates/
modules/
filters/
directives/
models/
services/
controllers/
```

В `app/assets/javascripts/application.js` добавьте `require` для Angular, помощника шаблона и структуры файла Angular app. Как это:

```
//= 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
```

---

## Шаг 5: Загрузочное угловое приложение

**Создайте** `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=csrf-token]').attr('content') ]) @app.run(-> console.log 'angular app running'
)
```

**Создайте модуль** `app/assets/javascripts/angular-app/modules/example.js.coffee.erb` **B**

`app/assets/javascripts/angular-app/modules/example.js.coffee.erb` :

```
@exampleApp = angular.module('app.exampleApp', [ # additional dependencies here ])
.run(-> console.log 'exampleApp running' )
```

**Создайте угловой контроллер для этого приложения в** `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" ])
```

**Теперь добавьте маршрут к Rails, чтобы передать управление Angular. В** `config/routes.rb` :

```
Rails.application.routes.draw do get 'example' => 'example#index' end
```

**Создайте контроллер Rails для ответа на этот маршрут:**

```
rails g controller Example
```

**В** `app/controllers/example_controller.rb` :

```
class ExampleController < ApplicationController
  def index
    end
end
```

**В представлении нам нужно указать, какое приложение Angular и какой угловой контроллер будет управлять этой страницей. Итак, в** `app/views/example/index.html.erb` :

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>

  <p>Value from ExampleCtrl:</p>
  <p>{{ exampleValue }}</p>

</div>
```

Чтобы просмотреть приложение, запустите сервер Rails и посетите <http://localhost:3000/example> .

Прочитайте **Настройка углового с рельсами онлайн**: <https://riptutorial.com/ru/ruby-on-rails/topic/3902/настройка-углового-с-рельсами>

# глава 57: Неверная маршрутизация

## Examples

### 1. Использование мелкой

Один из способов избежать глубокой вложенности (как рекомендовано выше) состоит в том, чтобы генерировать действия коллекции, охваченные под родительским элементом, чтобы получить представление об иерархии, но не вставлять действия элемента. Другими словами, чтобы создавать маршруты с минимальным объемом информации для уникальной идентификации ресурса, например:

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

Неглубокий метод DSL создает область, внутри которой каждое гнездо неглубоко. Это создает те же маршруты, что и в предыдущем примере:

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

Существуют два варианта возможностей для настройки мелких маршрутов. `: shallow_path` префикс пути элемента с указанным параметром:

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

Используйте команду Rake для получения сгенерированных маршрутов, как указано ниже:

```
rake routes
```

Прочитайте Неверная маршрутизация онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/7775/неверная-маршрутизация>

# глава 58: Обновление рельсов

## Examples

### Обновление с Rails 4.2 до Rails 5.0

*Примечание. Прежде чем обновлять приложение Rails, обязательно сохраните код в системе управления версиями, например Git.*

Чтобы перейти с Rails 4.2 на Rails 5.0, вы должны использовать Ruby 2.2.2 или новее. После обновления вашей версии Ruby, если необходимо, перейдите в свой Gemfile и измените строку:

```
gem 'rails', '4.2.X'
```

чтобы:

```
gem 'rails', '~> 5.0.0'
```

и в командной строке:

```
$ bundle update
```

Теперь запустите задачу обновления с помощью команды:

```
$ rake rails:update
```

Это поможет вам обновить файлы конфигурации. Вам будет предложено перезаписать файлы, и у вас есть несколько вариантов ввода:

- Y - да, перезаписать
- n - нет, не перезаписывать
- a - все, перезаписать это и все остальные
- q - выйти, прервать
- d - diff, показать различия между старым и новым
- h - помощь

Как правило, вы должны проверить различия между старыми и новыми файлами, чтобы убедиться, что вы не получаете никаких нежелательных изменений.

Модели Rails 5.0 `ActiveRecord` наследуются от `ApplicationRecord`, а не `ActiveRecord::Base`.

`ApplicationRecord` - это суперкласс для всех моделей, аналогичный тому, как

`ApplicationController` является суперклассом для контроллеров. Чтобы учесть этот новый

способ обработки моделей, вы должны создать файл в своей папке `app/models/` folder, называемый `application_record.rb` а затем отредактировать содержимое этого файла:

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0 также обрабатывает обратные вызовы несколько иначе. Обратные вызовы, возвращающие `false`, не остановят цепочку обратного вызова, а это означает, что последующие обратные вызовы будут выполняться, в отличие от Rails 4.2. При обновлении поведение Rails 4.2 останется, хотя вы можете переключиться на поведение Rails 5.0, добавив:

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

в файл `config/application.rb`. Вы можете явно заблокировать цепочку обратного вызова, вызвав `throw(:abort)`.

В Rails 5.0 `ApplicationJob` наследует от `ApplicationJob`, а не `ActiveJob::Base` как в Rails 4.2. Чтобы перейти на Rails 5.0, создайте файл `application_job.rb` в папке `app/jobs/`. Измените содержимое этого файла:

```
class ApplicationJob < ActiveJob::Base
end
```

Затем вы должны изменить все свои задания, чтобы наследовать от `ApplicationJob` а не `ActiveJob::Base`.

Одно из самых больших изменений Rails 5.0 не требует каких-либо изменений кода, но изменит способ использования командной строки с вашими приложениями Rails. Вы сможете использовать `bin/rails` или просто `rails` для запуска задач и тестов. Например, вместо использования `$ rake db:migrate` теперь вы можете сделать `$ rails db:migrate`. Если вы запустите `$ bin/rails`, вы можете просмотреть все доступные команды. Обратите внимание, что многие задачи, которые теперь могут выполняться с `bin/rails` все еще работают с использованием `rake`.

Прочитайте Обновление рельсов онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3496/обновление-рельсов>



---

# глава 59: Организация классов

## замечания

Это похоже на простое дело, но когда вы начинаете занятия в классе, вы будете благодарны, что потратили время на их организацию.

## Examples

### Модельный класс

```
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
```

Модели обычно отвечают за:

- установление отношений
- проверка достоверности данных
- обеспечение доступа к данным по областям и методам
- Выполнение действий по сохранению данных.

На самом высоком уровне модели описывают концепции домена и управляют их

персистентностью.

## Класс обслуживания

Контроллер является точкой входа в наше приложение. Однако это не единственная возможная точка входа. Я хотел бы получить доступ к моей логике:

- Рейк-задачи
- фоновые задания
- приставка
- тесты

Если я введу свою логику в контроллер, она не будет доступна из всех этих мест. Итак, давайте попробуем «тощий контроллер, толстая модель» и переместим логику к модели. Но какой? Если данный фрагмент логики включает модели `User`, `Cart` и `Product` - где он должен жить?

Класс, который наследует от `ActiveRecord::Base` уже имеет много обязанностей. Он обрабатывает интерфейс запроса, ассоциации и проверки. Если вы добавите еще больше кода в свою модель, он быстро станет недостижимым беспорядком с сотнями общественных методов.

Служба - это обычный объект Ruby. Его класс не должен наследовать ни от какого конкретного класса. Его имя - это глагольная фраза, например `CreateUserAccount` а не `UserCreation` или `UserCreationService`. Он живет в каталоге приложений / сервисов. Вы должны создать этот каталог самостоятельно, но Rails будет автоматически загружать классы внутри вас.

### Объект службы делает одну вещь

Объект службы (объект метода) выполняет одно действие. Он выполняет бизнес-логику для выполнения этого действия. Вот пример:

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

Ниже приведены три конвенции:

Услуги идут под `app/services` directory. Я рекомендую вам использовать подкаталоги для бизнес-логики. Например:

- Файл `app/services/invite/accept.rb` будет определять `Invite::Accept` пока

app/services/invite/create.rb определит Invite::Create

- Услуги начинаются с глагола (и не заканчиваются Сервисом): `ApproveTransaction`, `SendTestNewsletter`, `ImportUsersFromCsv`
- Услуги отвечают на метод `call`. Я обнаружил, что использование другого глагола делает его немного избыточным: `ApproveTransaction.approve()` не читается хорошо. Кроме `call` метод `call` является де-факто-методом для объектов `lambda`, `procs` и методов.

## Выгоды

### *Объекты службы показывают, что делает мое приложение*

Я могу просто взглянуть на каталог служб, чтобы узнать, что делает мое приложение:

`ApproveTransaction`, `CancelTransaction`, `BlockAccount`, `SendTransactionApprovalReminder` ...

Быстрый взгляд на объект службы, и я знаю, что такое бизнес-логика. Мне не нужно проходить через контроллеры, обратные вызовы модели `ActiveRecord` и наблюдатели, чтобы понять, что подразумевает «одобрение транзакции».

### *Модифицированные модели и контроллеры*

Контроллеры превращают запрос (`params`, `session`, `cookies`) в аргументы, передают их службе и перенаправляют или обрабатывают в соответствии с ответом службы.

```
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
```

Модели имеют дело только с ассоциациями, областями, валидациями и настойчивостью.

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

Это упрощает тестирование и обслуживание моделей и контроллеров!

### **Когда использовать класс обслуживания**

Достижение для объектов службы, когда действие соответствует одному или нескольким

из следующих критериев:

- Действие является сложным (например, закрытие книг в конце отчетного периода)
- Действие распространяется на несколько моделей (например, покупка электронной коммерции с использованием объектов Order, CreditCard и Customer)
- Действие взаимодействует с внешней службой (например, публикация в социальных сетях)
- Действие не является основной проблемой базовой модели (например, сбрасывание устаревших данных через определенный период времени).
- Существует несколько способов выполнения действия (например, аутентификация с помощью токена доступа или пароля).

## ИСТОЧНИКИ

[Блог Адама Недзельского](#)

[Блог Brew House](#)

[Блог о климате Code](#)

Прочитайте [Организация классов онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/7623/](https://riptutorial.com/ru/ruby-on-rails/topic/7623/)  
[организация-классов](#)

---

# глава 60: отладка

## Examples

### Отладочное приложение Rails

Чтобы иметь возможность отлаживать приложение, очень важно понять поток логики и данных приложения. Это помогает решать логические ошибки и повышает ценность опыта программирования и качества кода. Двумя популярными камнями для отладки являются [отладчик](#) (для ruby 1.9.2 и 1.9.3) и [byebug](#) (для ruby >= 2.x).

Для отладки файлов `.rb` выполните следующие действия:

1. Добавить `debugger` или `byebug` в группу `development` Gemfile
2. Запустить `bundle install`
3. Добавить `debugger` или `byebug` в качестве точки останова
4. Запустить код или сделать запрос
5. См. Журнал сервера rails, остановленный в указанной точке останова
6. На этом этапе вы можете использовать свой серверный терминал так же, как `rails console` и проверить значения переменной и `params`
7. Для перехода к следующей инструкции введите `next` команду и нажмите клавишу `enter`
8. Для выхода из типа `c` и нажмите `enter`

Если вы хотите отлаживать файлы `.html.erb`, точка `.html.erb` будет добавлена как `<% debugger %>`

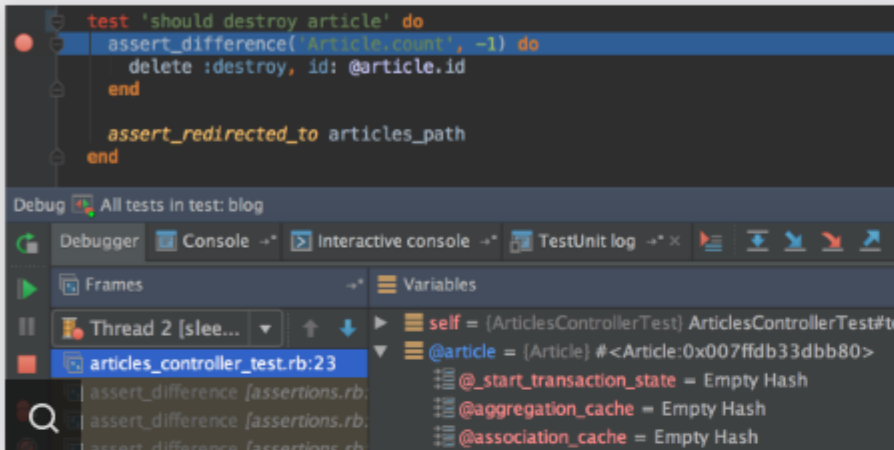
### Отладка в вашей среде IDE

Каждая хорошая [среда IDE](#) предоставляет [графический интерфейс](#) для интерактивной отладки приложений Ruby (и, таким образом, Rails), где вы можете добавлять точки останова, часы, автоматически приостанавливать выполнение исключения и позволяет выполнять выполнение кода поэтапно, по очереди.

Например, взгляните на одну из лучших Ruby IDE, функции отладки RubyMine на картинке

# 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.



## 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.

## 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.

## 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.

## Convenient user interface

- Look under the hood of any object with the Variables and Watches views.
- The UI is fully customizable: you can select toolbar commands, project code while stepping through it, and so on.
- The debugger UI is also tightly integrated with the IDE, so you can navigate between the debugger and the code editor, etc.
- You also get the complete set of debugging views.

## Debugging JavaScript

- RubyMine provides an advanced JavaScript debugger, which works with Google Chrome and Firefox.
- You can easily debug ECMAScript code running on RubyMine debugger's server.
- A full-featured debugger for Node.js, so you can debug apps running locally or remotely.

## Dedicated Watches

Track any number of expressions and variables in the current stack frame context. The Watches view is available through your debugging session.

## Remote debugging

As you connect to a remote host, you can use the mapping between the local source code and the remote debug processes can be launched.

## Отладка Ruby on Rails Быстро + советы для начинающих

**Отладка путем повышения исключений** гораздо проще, чем щурясь через `print` заявления журнала, и для большинства ошибок, его *намного быстрее*, чем открытие КРП отладчик, как `pry` или `byebug`. Эти инструменты не должны быть вашим первым шагом.

---

# Отладка Ruby / Rails быстро:

## 1. Быстрый метод: Поднимите `Exception` затем и `.inspect` его результат.

Самый *быстрый* способ отладки Ruby (особенно Rails) код для `raise` исключения по пути выполнения вашего кода при вызове `.inspect` на метод или объект (например, `foo`):

```
raise foo.inspect
```

В приведенном выше коде `raise` вызывает триггеры `Exception` которые *останавливают выполнение вашего кода*, и возвращает сообщение об ошибке, которое удобно содержит `.inspect` информацию об объекте / методе (т.е. `foo`) в строке, которую вы пытаетесь отлаживать.

Этот метод полезен для *быстрого* изучения объекта или метода (например, это `nil`?) И для немедленного подтверждения того, что строка кода даже вообще выполняется в рамках данного контекста.

## 2. Запасной: Используйте рубиновый *IRB* отладчик как

`byebug` **ИЛИ** `pry`

Только после того, как у вас есть информация о состоянии вашего кодов потока выполнения вы должны рассмотреть вопрос о переходе на рубин драгоценный камень *IRB* отладчик, как `pry` или `byebug`, где вы можете углубиться в состояние объектов в пределах вашего пути выполнения.

Чтобы использовать камень `byebug` для отладки в Rails:

1. Добавить `gem 'byebug'` внутри группы *разработки* в вашем *Gemfile*
2. Запустить `bundle install`
3. Затем, чтобы использовать, вставьте фразу `byebug` внутри пути выполнения кода, который вы хотите проверить.

Эта переменная `byebug` при ее `byebug` откроет рубиновый *IRB*-сеанс вашего кода, давая вам

прямой доступ к состоянию объектов, поскольку они находятся в этой точке выполнения кода.

Отладчики IRB, такие как Byebug, полезны для глубокого анализа состояния вашего кода по мере его выполнения. Тем не менее, они более трудоемкие процедуры по сравнению с повышением ошибок, поэтому в большинстве ситуаций они не должны быть вашим первым шагом.

---

## Общий совет для начинающих

Когда вы пытаетесь отладить проблему, рекомендуется всегда: **читать сообщение об ошибке @ @ \$ \$ Error (RTFM)**

Это означает, что вы читаете сообщения об ошибках *тщательно* и *полностью*, прежде чем действовать, чтобы вы *поняли, что он пытается сказать вам*. Когда вы отлаживаете ошибку, задайте следующие психические вопросы *в этом порядке* при чтении сообщения об ошибке:

1. Какой **класс** ссылается на ошибку? (т. е. есть ли у меня *правильный класс объекта или мой объект nil* ?)
2. Какой **метод** ссылается на ошибку? (т. е. является ли *их типом в методе, можно ли вызвать этот метод для этого типа / класса объекта*?)
3. Наконец, используя то, что я могу сделать из моих последних двух вопросов, какие **строки кода** следует исследовать? (помните: последняя строка кода в трассировке стека не обязательно там, где проблема лежит.)

В трассировке стека обратите особое внимание на строки кода, которые приходят из вашего проекта (например, строки, начинающиеся с `app/...` если вы используете Rails). В 99% случаев проблема связана с вашим собственным кодом.

---

Чтобы проиллюстрировать, почему интерпретация *в этом порядке* важна ...

**Например, сообщение об ошибке Ruby, которое путает многих новичков:**

Вы выполняете код, который в какой-то момент выполняется как таковой:

```
@foo = Foo.new
...
@foo.bar
```

и вы получите сообщение об ошибке:

```
undefined method "bar" for Nil:nilClass
```



Начинающие видят эту ошибку и считают, что проблема в том, что `bar` методов не определена. **Это не.** В этой ошибке реальная часть имеет значение:

```
for Nil:nilClass
```

`for Nil:nilClass` означает, что `@foo` - это **Nil!** `@foo` не является переменной экземпляра `Foo`! У вас есть объект, который является `Nil`. Когда вы видите эту ошибку, просто `ruby` пытается сказать вам, что `bar` методов не существует для объектов класса `Nil`. (ну, так как мы пытаемся использовать метод для объекта класса `Foo` не `Nil`).

К сожалению, из-за того, как эта ошибка записывается (не `undefined method "bar" for Nil:nilClass`) его легко получить обмануты, думая, эту ошибку нужно делать с `bar` является `undefined`. Когда вы не читаете внимательно, эта ошибка заставляет новичков ошибочно переходить к деталям метода `bar` на `Foo`, полностью упуская часть ошибки, которая подсказывает, что объект имеет неправильный класс (в данном случае: `nil`). Это ошибка, которую легко избежать, читая сообщения об ошибках полностью.

## Резюме:

Всегда внимательно **прочитайте все сообщение об ошибке** перед началом любой отладки. Это означает: всегда проверяйте тип **класса** объекта в сообщении об ошибке *сначала*, а затем его **методы**, прежде чем приступить к любому стеку или строке кода, где, по вашему мнению, может возникнуть ошибка. Эти 5 секунд могут сэкономить вам 5 часов разочарования.

**tl; dr:** Не косоглазие в журналах печати: вместо этого выведите исключения. Избегайте кроличьих отверстий, тщательно читая ошибки перед отладкой.

## Отладка приложения `ruby-on-rails` с помощью `pry`

`pry` - мощный инструмент, который можно использовать для отладки любого рубинового приложения. Настройка приложения `ruby-on-rails` с помощью этого драгоценного камня очень проста и понятна.

### Настроить

Чтобы начать отладку приложения с помощью `pry`

- Добавьте `gem 'pry'` в `Gemfile` приложения и `Gemfile` его

```
group :development, :test do
  gem 'pry'
end
```

- Перейдите в корневой каталог приложения на консоль терминала и выполните `bundle install`. Вы все можете начать использовать его в любом месте приложения.

## использование

Использование `pry` в вашем приложении просто включает `binding.pry` на контрольных точках, которые вы хотите проверить во время отладки. Вы можете добавить `binding.pry` точку останова в любом месте вашего приложения, которое интерпретируется рублиновым переводчиком (любое приложение / контроллеры, приложение / модель, приложение / файлы просмотра)

### i) Отладка контроллера

*приложение / контроллеры / users\_controller.rb*

```
class UsersController < ApplicationController
  def show
    user_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
```

В этом примере сервер rails приостанавливается с помощью штыревой консоли в точке разрыва при попытке посетить маршрутизацию страницы, чтобы `show` действие на `UserController`. Вы можете проверить объект `params` и сделать запрос ActiveRecord на модели `User` с этой точки останова

### ii) Отладка представления

*приложение / просмотров / пользователей / 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"
```

В этом примере точка прерывания приостанавливается с помощью консоли `pry`, когда страница `users/show` предварительно скомпилирована на сервере rails, прежде чем

отправлять ее обратно в браузер клиента. Эта точка прерывания позволяет отлаживать правильность `@user.logged_in?` когда это плохо.

## ii) Отладка модели

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

В этом примере точка прерывания может использоваться для отладки имени `full_name` экземпляра модели `User` `full_name` когда этот метод вызывается из любого места приложения.

В заключение, pry - мощный инструмент для отладки для приложения rails с простой настройкой и простой инструкцией по отладке. Попробуй.

Прочитайте отладка онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3877/отладка>

# глава 61: Проверка ActiveRecord

## Examples

### Проверка количественной характеристики атрибута

Эта проверка ограничивает вставку только числовых значений.

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

Кроме того `:only_integer`, этот помощник также принимает следующие параметры для добавления ограничений к допустимым значениям:

- `:greater_than` - указывает, что значение должно быть больше `:greater_than` значения. Сообщение об ошибке по умолчанию для этого параметра «должно быть больше% {count}».
- `:greater_than_or_equal_to` - Указывает, что значение должно быть больше или равно `:greater_than_or_equal_to` значению. Сообщение об ошибке по умолчанию для этой опции «должно быть больше или равно% {count}».
- `:equal_to` - Указывает, что значение должно быть равно `:equal_to` значению. Сообщение об ошибке по умолчанию для этой опции «должно быть равно% {count}».
- `:less_than` - Указывает, что значение должно быть меньше `:less_than` значения. Сообщение об ошибке по умолчанию для этого параметра «должно быть меньше% {count}».
- `:less_than_or_equal_to` - Указывает, что значение должно быть меньше или равно `:less_than_or_equal_to` значению. Сообщение об ошибке по умолчанию для этого параметра «должно быть меньше или равно% {count}».
- `:other_than` - Указывает, что значение должно быть иным, чем заданное значение. Сообщение об ошибке по умолчанию для этой опции «должно быть отличным от% {count}».
- `:odd` - указывает, что значение должно быть нечетным числом, если установлено значение true. Сообщение об ошибке по умолчанию для этой опции «должно быть нечетным».
- `:even` - Указывает, что значение должно быть четным числом, если установлено значение true. Сообщение об ошибке по умолчанию для этой опции «должно быть четным».

По умолчанию численное значение не допускает значений nil. Вы можете использовать `allow_nil: true`, чтобы разрешить это.

## Проверка уникальности атрибута

Этот помощник подтверждает, что значение атрибута уникально непосредственно перед тем, как объект будет сохранен.

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

Существует опция `:scope` которую вы можете использовать для указания одного или нескольких атрибутов, которые используются для ограничения проверки уникальности:

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

Также существует опция `:case_sensitive` которую вы можете использовать для определения того, будет ли ограничение уникальности чувствительным к регистру или нет. Эта опция по умолчанию имеет значение `true`.

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

## Проверка наличия атрибута

Этот помощник проверяет, что указанные атрибуты не пусты.

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

Вы можете использовать помощник `absence` чтобы проверить, что указанные атрибуты отсутствуют. Он использует `present?` метод для проверки нулевого или пустого значений.

```
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

**Примечание.** Если атрибут является `boolean`, вы не можете использовать обычную проверку присутствия (атрибут недействителен для `false` значения). Вы можете сделать это, используя проверку включения:

```
validates :attribute, inclusion: [true, false]
```

## Пропуск проверки

Используйте следующие методы, если вы хотите пропустить проверки. Эти методы сохранят объект в базе данных, даже если он недействителен.

- уменьшаем!
- `decrement_counter`
- увеличиваем!
- `increment_counter`
- переключения!
- потрогать
- обновить все
- `update_attribute`
- `update_column`
- `update_columns`
- `update_counters`

Вы также можете пропустить проверку при сохранении, пройдя `validate` в качестве аргумента для `save`

```
User.save(validate: false)
```

## Проверка длины атрибута

```
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
```

Возможные варианты ограничения длины:

- `:minimum` - атрибут не может иметь меньше заданной длины.
- `:maximum` - атрибут не может иметь больше указанной длины.
- `:in` (или `:within`) - длина атрибута должна быть включена в заданный интервал. Значение для этой опции должно быть диапазоном.
- `:is` - Длина атрибута должна быть равна заданному значению.

## Подтверждение группировки

Иногда полезно иметь несколько проверок, используя одно условие. Его можно легко достичь, используя `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

```

Все проверки внутри блока `with_options` автоматически передают условие, если:: `is_admin?`

## Пользовательские проверки

Вы можете добавить свои собственные проверки, добавив новые классы, наследующие от `ActiveModel::Validator` или из `ActiveModel::EachValidator`. Оба метода похожи, но они работают несколько иначе:

`ActiveModel::Validator` **И** `validates_with`

Внедрите метод `validate` который принимает запись как аргумент и выполняет проверку на нем. Затем используйте `validates_with` с классом модели.

```

# 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` **И** `validate`

Если вы предпочитаете использовать свой новый валидатор с использованием общего метода `validate` на одном параметре, создайте класс, наследующий от `ActiveModel::EachValidator` и реализуйте метод `validate_each` который принимает три аргумента: `record`, `attribute` и `value`:

```

class EmailValidator < ActiveModel::EachValidator
  def validate_each(record, attribute, value)
    unless value =~ /\A([\s\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

```

Дополнительная информация о [направляющих Rails](#) .

## Проверяет формат атрибута

Подтвердить , что значение атрибута соответствует регулярному выражению , используя `format` и `with` опцией.

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

Вы также можете определить константу и установить ее значение в регулярное выражение и передать ее опции `with:` . Это может быть более удобным для действительно сложных регулярных выражений

```
PHONE_REGEX = /\A\(\d{3}\)\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

Сообщение об ошибке по умолчанию является `is invalid` . Это можно изменить с помощью опции `:message` .

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

Обратное также отвечает, и вы можете указать, что значение *не* должно соответствовать регулярному выражению с параметром `without:`

## Проверяет включение атрибута

Вы можете проверить, включено ли значение в массив с помощью `inclusion: helper` . Параметр `:in` и его псевдоним `:within` отображают набор допустимых значений.

```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                           Europe North America South America) }
end
```

Чтобы проверить, не включено ли значение в массив, используйте `exclusion:` хелпер

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

## Условная проверка

Иногда вам может потребоваться подтверждение записи только при определенных условиях.



```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

Если вы условно очень малы, вы можете использовать Proc:

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

Для отрицательных условий вы можете использовать, unless :

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

Вы также можете передать строку, которая будет выполнена через instance\_eval :

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

## Подтверждение атрибута

Вы должны использовать это, когда у вас есть два текстовых поля, которые должны получать точно такое же содержимое. Например, вы можете подтвердить адрес электронной почты или пароль. Эта проверка создает **виртуальный** атрибут, имя которого является именем поля, которое должно быть подтверждено добавлением `_confirmation`.

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

**Примечание.** Эта проверка выполняется только в том случае, если `email_confirmation` не равна нулю.

Чтобы потребовать подтверждения, обязательно добавьте проверку наличия для атрибута подтверждения.

```
class Person < ApplicationRecord
  validates :email, confirmation: true
  validates :email_confirmation, presence: true
end
```

[Источник](#)

## Использование: по опции

Опция `:on` позволяет указать, когда должна произойти проверка. Поведение по умолчанию для всех встроенных помощников проверки должно выполняться при сохранении (как при создании новой записи, так и при ее обновлении).

```
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
```

Прочитайте Проверка ActiveRecord онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/2105/проверка-activerecord>

---

# глава 62: Просмотры

## Examples

### Partials

Частичные шаблоны (частичные) - это способ разбить процесс рендеринга на более управляемые куски. Частицы позволяют извлекать фрагменты кода из ваших шаблонов для разделения файлов, а также повторно использовать их во всех шаблонах.

Чтобы *создать* частичный, создайте новый файл, начинающийся с `_form.html.erb` подчеркивания: `_form.html.erb`

Чтобы *сделать* частичное как часть представления, используйте метод рендеринга в представлении: `<%= render "form" %>`

- Обратите внимание, что подчеркивание не учитывается при рендеринге
- Частичный должен быть отображен с использованием его пути, если он находится в другой папке

Чтобы *передать* переменную в частичную как локальную переменную, используйте это обозначение:

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

Частицы также полезны, когда вам нужно *повторно использовать* точно такой же код ( **концепция DRY** ).

Например, для повторного использования кода `<head>` , создайте частичное имя `_html_header.html.erb` , введите свой `<head>` код, который нужно повторно использовать, и отрисуйте частичный, когда это необходимо: `<%= render 'html_header' %>` .

---

## Частицы объектов

Объекты, которые отвечают на `to_partial_path` также могут быть отображены, как в `<%= render @post %>` . По умолчанию для моделей ActiveRecord это будет что-то вроде `posts/post` , поэтому, фактически, рендеринга `@post` , будут отображаться файлы `views/posts/_post.html.erb` .

Локальная именованная `post` будет автоматически назначена. В конце концов, `<%= render @post %>` - это короткая рука для `<%= render 'posts/post', post: @post %>` .

`to_partial_path` могут быть представлены коллекции объектов, которые отвечают на `to_partial_path` , например `<%= render @posts %>` . Каждый элемент будет отображаться

последовательно.

---

## Глобальные Частицы

Чтобы создать глобальный фрагмент, который можно использовать в любом месте без ссылки на его точный путь, частичное должно быть расположено в пути `views/application`. Предыдущий пример был изменен ниже, чтобы проиллюстрировать эту функцию.

Например, это путь к глобальному частичному `app/views/application/_html_header.html.erb`:

Чтобы сделать это глобальное частичное в любом месте, используйте `<%= render 'html_header' %>`

### AssetTagHelper

Чтобы рельсы автоматически и правильно связывали ресурсы (css / js / images), в большинстве случаев вы хотите использовать встроенные помощники. ( [Официальная документация](#) )

---

## Помощники изображения

### *image\_path*

Это возвращает путь к объекту изображения в `app/assets/images`.

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

### *URL изображения*

Это возвращает полный URL-адрес ресурса изображения в `app/assets/images`.

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

### **IMAGE\_TAG**

Используйте этот помощник, если вы хотите включить `<img src="" />` -tag с исходным набором.

```
image_tag("icon.png") # => 
```

# Помощники JavaScript

## *javascript\_include\_tag*

Если вы хотите включить JavaScript-файл в свой вид.

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

## *javascript\_path*

Это возвращает путь к вашему JavaScript-файлу.

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

## *javascript\_url*

Это возвращает полный URL вашего JavaScript-файла.

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

---

# Помощники стилей

## *stylesheet\_link\_tag*

Если вы хотите включить CSS-файл в свой вид.

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

## *stylesheet\_path*

Это возвращает путь к вашему ресурсу таблицы стилей.

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

## *stylesheet\_url*

Это возвращает полный URL-адрес вашего ресурса таблицы стилей.

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

# Пример использования

При создании нового приложения rails у вас автоматически будут два этих помощника в `app/views/layouts/application.html.erb`

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

Эти результаты:

```
// CSS
<link rel="stylesheet" media="all" href="/assets/application.self-
e19d4b856cacba4f6fb0e5aa82alba9aa4ad616f0213a1259650b281d9cf6b20.css?body=1" data-turbolinks-
track="reload" />
// JavaScript
<script src="/assets/application.self-
619d9bf310b8eb258c67de7af745cafbf2a98f6d4c7bb6db8e1b00aed89eb0b1.js?body=1" data-turbolinks-
track="reload"></script>
```

## Состав

Поскольку Rails следует за шаблоном **MVC** `Views`, где ваши «шаблоны» предназначены для ваших действий.

Допустим, у вас есть контроллер `articles_controller.rb`. Для этого контроллера у вас будет папка в виде под названием `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
|
|-- [...]
```

Эта структура позволяет вам иметь папку для каждого контроллера. При вызове действия в контроллере соответствующее представление будет отображаться автоматически.

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

// show.html.erb
```

```
<h1>My show view</h1>
```

## Заменить HTML-код в Views

Если вы когда-либо хотели определить содержимое html, которое будет напечатано на странице во время выполнения, тогда для этого рельсы имеют очень хорошее решение. У этого есть что-то, называемое **content\_for**, которое позволяет нам передать блок в представление рельсов. Пожалуйста, проверьте приведенный ниже пример,

### Объявить content\_for

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

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

## HAML - альтернативный способ использования в ваших представлениях

HAML (язык разметки HTML-абстракции) - это прекрасный и элегантный способ описания и дизайна HTML ваших взглядов. Вместо меток открытия и закрытия HAML использует отступы для структуры ваших страниц. В принципе, если что-то должно быть помещено в другой элемент, вы просто отступаете его, используя одну вкладку. В HAML важны вкладки и пробелы, поэтому убедитесь, что вы всегда используете одинаковое количество вкладок.

### Примеры:

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

### И в HAML:

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

Видите ли, структура макета намного яснее, чем использование HTML и ERB.

### Монтаж

Просто установите драгоценный камень, используя

```
gem install haml
```

и добавьте драгоценный камень в Gemfile

```
gem "haml"
```

Для использования HAML вместо HTML / ERB просто замените расширения файлов ваших представлений от `something.html.erb` на `something.html.haml`.

## Быстрые подсказки

Общие элементы, такие как `div`s, могут быть написаны коротким образом

### HTML

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

### HAML

```
%div.myclass
```

### HAML, стенография

```
.myclass
```

## Атрибуты

### HTML

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

### HAML

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

## Вставка кода `guby`

Вы можете вставить код `guby` с знаками `=` и `-`.

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

Код, начинающийся с `=`, будет выполнен и встроен в документ.

Код, начинающийся с `-` будет выполнен, но не вставлен в документ.



## Полная документация

HAML очень легко начать, но также очень сложно, поэтому я рекомендую [прочитать документацию](#) .

Прочитайте Просмотры онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/850/просмотры>

# глава 63: Развертывание приложения Rails на Heroku

## Examples

### Развертывание приложения

Убедитесь, что вы находитесь в каталоге, который содержит ваше приложение Rails, а затем создайте приложение на Heroku.

```
$ heroku create example
Creating ▯ example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

Первый URL-адрес выхода, <http://example.herokuapp.com>, - это местоположение, в котором доступно приложение. Второй URL-адрес, `git@heroku.com: example.git`, является удаленным URL-адресом репозитория git.

Эта команда должна использоваться только в инициализированном репозитории git. Команда `create heroku` автоматически добавляет `git remote` с именем «heroku», указывающий на этот URL.

Аргумент имени приложения («пример») является необязательным. Если имя приложения не указано, будет создано случайное имя. Поскольку имена приложений Heroku находятся в глобальном пространстве имен, вы можете ожидать, что общие имена, такие как «блог» или «вики», уже будут приняты. Часто проще начать с имени по умолчанию и позже переименовать приложение.

Затем разверните свой код:

```
$ 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:      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
remote:      the lockfile. We suggest you upgrade to the latest version of Bundler by running `gem install
remote:      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-
1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js
remote:      I, [2016-07-08T17:08:57.046951 #1222] INFO -- : Writing
/tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.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:      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

```

Если вы используете базу данных в своем приложении, вам необходимо вручную выполнить миграцию базы данных, выполнив:

```
$ heroku run rake db:migrate
```

Любые команды после `heroku run` будут исполняться на `heroku run` Heroku. Вы можете получить интерактивный сеанс оболочки, выполнив:

```
$ heroku run bash
```

Убедитесь, что у вас есть один динамический тип веб-процесса:

```
$ heroku ps:scale web=1
```

Команда `heroku ps` перечисляет текущие динамики вашего приложения:

```
$ 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)
```

Теперь вы можете посетить приложение в нашем браузере с `heroku open`.

```
$ heroku open
```

Heroku предоставляет веб-URL по умолчанию в домене `herokuapp.com`. Когда вы будете готовы к расширению для производства, вы можете добавить свой собственный домен.

## Управление производственными и промежуточными средами для Heroku

Каждое приложение Heroku работает как минимум в двух средах: на Heroku (мы будем называть это производством) и на вашей локальной машине (разработка). Если в приложении работает более одного человека, у вас есть несколько сред разработки - по одному на машину. Обычно у каждого разработчика также есть тестовая среда для запуска тестов. К сожалению, этот подход разрушается, поскольку среда становится менее похожей. Например, Windows и Mac предоставляют различные среды, чем стек Linux на Heroku, поэтому вы не всегда можете быть уверены, что код, который работает в вашей локальной среде разработки, будет работать одинаково, когда вы развертываете его на производство.

Решение состоит в том, чтобы иметь промежуточную среду, которая настолько же похожа на производство, насколько это возможно. Этого можно достичь, создав второе приложение Heroku, на котором размещено ваше промежуточное приложение. С помощью этапа вы можете проверить свой код в настройке, подобной производству, прежде чем он повлияет на ваших реальных пользователей.

### Начиная с нуля

Предположим, у вас есть приложение, запущенное на вашем локальном компьютере, и вы готовы нажать его на Heroku. Нам нужно создать как удаленные среды, так и постановку. Чтобы привыкнуть к тому, чтобы сначала продвигаться вперед, мы начнем с этого:

```
$ heroku create --remote staging
Creating strong-river-216.... done
http://strong-river-216.herokuapp.com/ | https://git.heroku.com/strong-river-216.git
Git remote staging added
```

По умолчанию CLI героя создает проекты с удаленным героем `git`. Здесь мы указываем другое имя с флагом `-remote`, поэтому нажатие кода на Heroku и запуск команд против приложения выглядят немного иначе, чем обычный `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
```

После того, как ваше промежуточное приложение работает правильно, вы можете создать свое производственное приложение:

```
$ heroku create --remote production
Creating fierce-ice-327.... done
http://fierce-ice-327.herokuapp.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
```

И с этим у вас есть тот же код, который работает как два отдельных приложения Heroku - одна постановка и одна постановка, настроенная одинаково. Просто помните, что вам нужно будет указать, какое приложение будет работать на вашей повседневной работе. Вы можете использовать флаг '--remote' или использовать конфигурацию git для указания приложения по умолчанию.

Прочитайте [Развертывание приложения Rails на Heroku онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/4485/развертывание-приложения-rails-на-heroku):

<https://riptutorial.com/ru/ruby-on-rails/topic/4485/развертывание-приложения-rails-на-heroku>

---

# глава 64: Рамки Rails на протяжении многих лет

## Вступление

Когда вы новичок в Rails и работаете с устаревшими приложениями Rails, может возникнуть недоумение понять, какая структура была введена, когда. Этот раздел предназначен для *окончательного* списка всех фреймворков в версиях Rails.

## Examples

### Как найти, какие рамки доступны в текущей версии Rails?

Использовать

```
config.frameworks
```

чтобы получить массив `Symbol` которые представляют каждую структуру.

### Версии Rails в Rails 1.x

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- Railties

### Рамки Rails в Rails 2.x

- ActionMailer
- ActionPack
- ActiveRecord
- ActiveResource ( *ActiveWebService* был заменен на *ActiveResource*, и с этим Rails переместился с SOAP на REST по умолчанию )
- ActiveSupport
- Railties

### Рамки Rails в Rails 3.x

- ActionMailer
- ActionPack

- ActiveModel
- ActiveRecord
- ActiveResource
- ActiveSupport
- Railties

Прочитайте Рамки Rails на протяжении многих лет онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/8107/рамки-rails-на-протяжении-многих-лет>



---

# глава 65: Реагировать с рельсами, используя камень с ребрами

## Examples

### Реагировать на установку Rails с использованием rails\_react gem

Добавьте ретрансляционные рельсы в ваш Gemfile:

```
gem 'react-rails'
```

И установите:

```
bundle install
```

Затем запустите сценарий установки:

```
rails g react:install
```

Это будет:

создайте файл манифеста components.js и каталог app / assets / javascripts / components /, где вы поместите свои компоненты в свое приложение application.js:

```
//= require react  
//= require react_ujs  
//= require components
```

## Использование response\_rails в вашем приложении

### React.js строит

Вы можете выбрать, какие сборки React.js (разработка, производство, с или без надстроек) для обслуживания в каждой среде, добавив конфигурацию. Вот настройки по умолчанию:

```
# 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
```

Чтобы включить надстройки, используйте эту конфигурацию:

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

После перезапуска сервера Rails `// = require response` предоставит сборку React.js, которая была задана конфигурациями.

Реактивные рельсы предлагают несколько других вариантов для версий и сборок React.js. См. `VERSIONS.md` для получения дополнительной информации об использовании драгоценного камня с реактивными источниками или о том, как использовать его в своих копиях React.js.

## JSX

После установки реактивных рельсов перезагрузите сервер. Теперь файлы `.js.jsx` будут преобразованы в конвейер активов.

### Параметры BabelTransformer

Вы можете использовать трансформаторы Babel и пользовательские плагины, а также передавать параметры транспилерам babel, добавляя следующие конфигурации:

```
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]
}
```

Под капотом [реактивные](#) рельсы используют [транспилер ruby-babel-transpiler](#) для трансформации.

## Рендеринг и монтаж

(`react_component`) `react-rails` включает в себя помощник вида (`react_component`) и ненавязчивый JavaScript-драйвер (`react_ujs`), которые работают вместе, чтобы помещать компоненты React на страницу. Вы должны потребовать драйвер UJS в манифесте после реагирования (и после турбовинтовых, если вы используете Turbolinks).

Помощник вида помещает `div` на страницу с запрошенным классом компонентов и реквизитами. Например:

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
```

```
props="{&quot;name&quot;:&quot;John&quot;}"></div>
```

При загрузке страницы драйвер `response_ujs` сканирует страницу и монтирует компоненты с помощью реквизитов для обработки данных и обработки данных.

Если присутствуют `Turbolinks`, компоненты устанавливаются на странице: изменение события и размонтирование на странице: перед выгрузкой. `Turbolinks` = 2.4.0 рекомендуется, поскольку он предоставляет лучшие события.

В случае вызовов Ajax установка UJS может запускаться вручную, вызывая из javascript:

`ReactRailsUJS.mountComponents ()` Подпись помощника вида:

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name` - это строка, которая называет глобально доступный класс компонентов. Он может иметь точки (например, «`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` могут включать в себя: `tag`: использовать элемент, отличный от `div`, для встраивания обработчиков данных и реагирования на данные. `prerender: true` для рендеринга компонента на сервере. `**other` Любые другие аргументы (например, `class :`, `id :`) передаются в `content_tag`.

Прочитайте [Реагировать с рельсами, используя камень с ребрами онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/7032/реагировать-с-рельсами-используя-камень-с-ребрами-онлайн):

<https://riptutorial.com/ru/ruby-on-rails/topic/7032/реагировать-с-рельсами-используя-камень-с-ребрами>

# глава 66: Рекомендации по Rails

## Examples

### Не повторяйте себя (DRY)

Чтобы поддерживать чистый код, Rails следует принципу DRY.

Он включает, когда это возможно, повторное использование как можно большего количества кода, а не дублирование аналогичного кода в нескольких местах (например, с использованием частичных). Это уменьшает количество *ошибок*, делает ваш код *чистым* и обеспечивает принцип *написания кода один раз* и повторное его использование. Также проще и эффективнее обновлять код в одном месте, чем обновлять несколько частей одного и того же кода. Таким образом, ваш код более модульный и надежный.

Также *Fat Model*, *Skinny Controller* DRY, потому что вы пишете код в своей модели, а в контроллере только вызов, например:

```
# 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
```

Это также помогает привести к структуре, управляемой API, где внутренние методы скрыты, а изменения достигаются с помощью передачи параметров по API.

### Контракт над конфигурацией

В Rails вы обнаружите, что просматриваете *контроллеры*, *представления* и *модели* для своей базы данных.

Чтобы уменьшить необходимость в большой конфигурации, Rails реализует правила, облегчающие работу с приложением. Вы можете определить свои собственные правила, но для начала (и для более позднего) это хорошая идея придерживаться соглашений, которые

предлагает Rails.

Эти соглашения ускоряют разработку, сохраняют ваш код кратким и читаемым и позволят вам легко перемещаться внутри вашего приложения.

Соглашения также снижают барьеры для входа для новичков. В Rails существует так много соглашений, что новичка даже не нужно знать, но может просто извлечь выгоду из невежества. Можно создавать отличные приложения, не зная, почему все так, как есть.

## Например

Если у вас есть таблица базы данных, называемая `orders` с `id` первичного ключа, соответствующая модель называется `order` а контроллер, который обрабатывает всю логику, называется `orders_controller`. Вид разделяется на разные действия: если у контроллера есть `new` действие и действие `edit`, есть также `new` и `edit` представление.

## Например

Чтобы создать приложение, вы просто запускаете `rails new app_name`. Это создаст примерно 70 файлов и папок, которые составляют инфраструктуру и основу для вашего приложения Rails.

Это включает:

- Папки для хранения ваших моделей (уровень базы данных), контроллеров и представлений
- Папки для проведения модульных тестов для вашего приложения
- Папки для хранения ваших сетевых ресурсов, таких как файлы Javascript и CSS
- Файлы по умолчанию для ответов HTTP 400 (т. Е. Файл не найден)
- Многие другие

## Жирная модель, тощий контроллер

«Fat Model, Skinny Controller» относится к тому, как М и С части MVC идеально работают вместе. А именно, любая логика, не связанная с ответом, должна идти в модели, в идеале, в хорошем, проверяемом методе. Между тем, «худой» контроллер - просто приятный интерфейс между представлением и моделью.

На практике это может потребовать множество различных типов рефакторинга, но все сводится к одной идее: путем перемещения любой логики, которая не связана с ответом модели (вместо контроллера), не только вы продвигаете повторное использование где это возможно, но вы также смогли проверить свой код вне контекста запроса.

Давайте посмотрим на простой пример. Скажем, у вас есть такой код:

```
def index
```

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

Вы можете изменить его на это:

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

Затем вы можете переместить логику в свою модель публикации, где она может выглядеть так:

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

## Остерегайтесь `default_scope`

ActiveRecord включает `default_scope`, чтобы автоматически поменять модель по умолчанию.

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

Вышеприведенный код будет обслуживать сообщения, которые уже опубликованы при выполнении любого запроса в модели.

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

Эта область, в то время как безвредная, имеет несколько скрытых побочных эффектов, которые вы, возможно, не захотите.

### `default_scope` и `order`

Поскольку вы объявили `order` в `default_scope`, вызывающий `order` в `Post` будет добавлен в качестве дополнительных заказов вместо переопределения значения по умолчанию.

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at"
DESC, "posts"."updated_at" DESC
```

Вероятно, это не то поведение, которое вы хотели; вы можете переопределить это, исключив сначала `order` из области

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at"
DESC
```

---

## default\_scope И инициализация модели

Как и в любом другом ActiveRecord::Relation, default\_scope изменит состояние по умолчанию, инициализированное им.

В приведенном выше примере Post имеет значение where(published: true) установленное по умолчанию, и поэтому новые модели из Post также будут установлены.

```
Post.new # => <Post published: true>
```

---

## unscoped

default\_scope можно номинально очистить, вызвав сначала не unscoped, но это также имеет побочные эффекты. Возьмем, к примеру, модель ИППП:

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

По умолчанию запросы к Post будут областями для type столбцов, содержащих 'Post'. Но unscoped очистит это вместе с вашим собственным default\_scope, поэтому, если вы используете unscoped вы должны помнить, чтобы учитывать это.

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

---

## unscoped И модельные ассоциации

Рассмотрите связь между Post и User

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

class User < ApplicationRecord
  has_many :posts
end
```

При получении отдельного User вы можете видеть связанные с ним сообщения:

```
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]]
```

Но вы хотите очистить `default_scope` от отношения `posts` , так что вы используете `unscoped`

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

Это уничтожает условие `user_id` а также `default_scope` .

## Пример использования для `default_scope`

Несмотря на все это, существуют ситуации, когда использование `default_scope` является оправданным.

Рассмотрим систему с несколькими арендаторами, где несколько поддоменов обслуживаются из одного приложения, но с изолированными данными. Одним из способов достижения этой изоляции является использование `default_scope` . Недостатки в других случаях могут стать причиной роста.

```
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
```

Все, что вам нужно сделать, это установить `Tenant.current_id` на что-то в начале запроса, и любая таблица, содержащая `tenant_id` , автоматически станет областью без дополнительного кода. Мгновенные записи автоматически наследуют идентификатор арендатора, с которым они были созданы.

Важное значение в этом случае состоит в том, что область задается один раз для каждого запроса и не изменяется. Единственные случаи , вы должны `unscoped` здесь особые случаи , такие как фон работники , которые работают за пределами области видимости запроса.

**Вам это не понадобится (YAGNI)**



Если вы можете сказать «YAGNI» (вам это не понадобится) о какой-либо функции, лучше не применять ее. Может быть много времени на разработку, сфокусированное на простоте. Реализация таких функций в любом случае может привести к проблемам:

---

## Проблемы

### переустройства

Если продукт сложнее, чем он должен быть, он переработан. Обычно эти «еще не используемые» функции никогда не будут использоваться по-своему, они были написаны и должны быть реорганизованы, если они когда-либо будут использоваться.

Преждевременные оптимизации, особенно оптимизация производительности, часто приводят к проектным решениям, которые в будущем будут ошибочными.

### Код Bloat

Code Bloat означает ненужный сложный код. Это может происходить, например, путем абстракции, избыточности или неправильного применения шаблонов проектирования. База кода становится трудно понять, запутанной и дорогой в обслуживании.

### Функция ползучести

Функция Сгеер относится к добавлению новых функций, которые выходят за рамки основных функций продукта и приводят к излишне высокой сложности продукта.

### Длительное время разработки

Время, которое может быть использовано для разработки необходимых функций, расходуется на разработку ненужных функций. Продукт требует больше времени для доставки.

---

## Решения

### KISS - Держите его простым, глупым

Согласно KISS, большинство систем работают лучше всего, если они разработаны просто. Простота должна быть основной целью проектирования, чтобы уменьшить сложность. Это может быть достигнуто, например, с помощью «Принципа единой ответственности».

# YAGNI - Вам это не понадобится

Меньше - больше. Подумайте о каждой функции, действительно ли это необходимо? Если вы можете думать о том, что это YAGNI, оставьте это. Лучше развивать его, когда это необходимо.

## Непрерывный рефакторинг

Продукт постоянно совершенствуется. С рефакторингом мы можем убедиться, что продукт выполняется в соответствии с лучшей практикой и не вырождается в работу патча.

### Объекты домена (больше нет моделей жира)

«Fat Model, Skinny Controller» - очень хороший первый шаг, но он плохо масштабируется, как только ваша кодовая база начинает расти.

Давайте подумаем о [единой ответственности](#) моделей. Какова единственная ответственность моделей? Это держать бизнес-логику? Должна ли она придерживаться логики, не связанной с ответом?

Нет. Его ответственность заключается в том, чтобы обрабатывать слой сохранения и его абстракцию.

Бизнес-логика, а также любая логика, не связанная с ответом, и логика, не связанная с сохранением, должны идти в объектах домена.

Объекты домена - это классы, предназначенные для одной ответственности в области проблемы. Пусть ваши классы « [Кричат их архитектуру](#) » для проблем, которые они решают.

На практике вы должны стремиться к тощим моделям, тощим представлениям и тощим контроллерам. Структура вашего решения не должна зависеть от структуры, которую вы выбираете.

### Например

Предположим, вы являетесь рынком, который взимает фиксированную комиссию в размере 15% для ваших клиентов через Stripe. Если вы взимаете фиксированную комиссию в размере 15%, это означает, что ваша комиссия изменяется в зависимости от суммы заказа, потому что сборы Strip 2.9% + 30 ¢.

Сумма, которую вы взимаете как комиссию, должна быть:  $amount * 0.15 - (amount * 0.029 + 0.30)$

.

Не пишите эту логику в модели:

```

# 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

```

Как только вы интегрируетесь с новым методом оплаты, вы не сможете масштабировать эту функциональность внутри этой модели.

Кроме того, как только вы начнете интегрировать больше бизнес-логики, ваш объект `Order` начнет терять **сплоченность** .

Предпочитают объекты домена, при этом расчет комиссии полностью абстрагируется от ответственности за сохраняющиеся приказы:

```

# 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
  end
end

```

```
+ 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
```

Использование объектов домена имеет следующие архитектурные преимущества:

- это очень просто для модульного тестирования, поскольку для создания объектов с логикой не требуются никакие приспособления или фабрики.
- работает со всем, что принимает `amount` сообщений.
- сохраняет каждый объект домена небольшим, с четко определенными обязанностями и с более высокой связностью.
- легко масштабируется с помощью новых способов оплаты путем [добавления, а не изменения](#) .
- прекращает тенденцию иметь постоянно растущий объект `User` в каждом приложении Ruby on Rails.

Я лично хотел бы поместить объекты домена в `lib` . Если вы это сделаете, не забудьте добавить его в `autoload_paths` :

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

Вы также можете предпочесть создавать объекты домена более ориентированными на действия, следуя шаблону `Command / Query` . В таком случае размещение этих объектов в `app/commands` может быть лучше, поскольку все подкаталоги `app` автоматически добавляются в путь автозагрузки.

Прочитайте [Рекомендации по Rails онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/1207/рекомендации-по-rails](https://riptutorial.com/ru/ruby-on-rails/topic/1207/рекомендации-по-rails)

# глава 67: Рельсы 5

## Examples

### Создание API Ruby on Rails 5

Чтобы создать новый API Rails 5, откройте терминал и выполните следующую команду:

```
rails new app_name --api
```

Будет создана следующая файловая структура:

```
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
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
```

Эта файловая структура будет создана в новой папке с именем `app_name` . Он содержит все активы и код, необходимые для запуска вашего проекта.

Войдите в папку и установите зависимости:

```
cd app_name
bundle install
```

Вы также должны запустить свою базу данных. Rails использует SQLite в качестве базы данных по умолчанию. Чтобы создать его, запустите:

```
rake db:setup
```

Теперь запустите приложение:

```
$ rails server
```

Когда вы открываете свой браузер по адресу `http://localhost:3000` , ваш блестящий новый

(пустой) API должен быть запущен!

## Как установить Ruby on Rails 5 на RVM

RVM - отличный инструмент для управления рубиновыми версиями и настройки рабочей среды.

Предполагая, что у вас уже установлен RVM, чтобы получить последнюю версию ruby, которая необходима для этих примеров, откройте терминал и запустите:

```
$ rvm get stable
$ rvm install ruby --latest
```

Проверьте свою рубиновую версию, запустив:

```
$ ruby -v
> ruby 2.3.0p0
```

Чтобы установить Rails 5, сначала создайте новый gemset, используя последнюю версию ruby, а затем установите рельсы:

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

Чтобы проверить версию рельсов, запустите:

```
$ rails -v
> Rails 5.0.0
```

Прочитайте Рельсы 5 онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/3019/рельсы-5>

---

# глава 68: Рельсы на докере

## Вступление

Этот учебник начнется с установленного Docker и с помощью приложения Rails

## Examples

### Докер и докер-сочиняют

Прежде всего, нам нужно будет создать наш `Dockerfile`. Хороший пример можно найти в этом [блоге](#) Nick Janetakis.

Этот код содержит сценарий, который будет выполнен на нашей докере-машине в момент запуска. По этой причине мы устанавливаем все необходимые библиотеки и заканчиваем началом Puma (RoR-dev-сервер)

```
# 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
```

Кроме того, мы будем использовать `docker-compose`, для этого мы создадим `docker-compose.yml`. Объяснение этого файла будет более учебным пособием для докеров, чем интеграция с Rails, и я не буду здесь останавливаться.

```
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
```

Просто с этими двумя файлами вам будет достаточно, чтобы запустить `docker-compose up` и разбудить ваш докер

Прочитайте Рельсы на докере онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/10933/рельсы-на-докере>

---

# глава 69: Сериализаторы активной модели

## Вступление

ActiveModelSerializers, или AMS для краткости, приносят «соглашение о конфигурации» в ваше поколение JSON. ActiveModelSerializers работают через два компонента: сериализаторы и адаптеры. Сериализаторы описывают, какие атрибуты и отношения должны быть сериализованы. Адаптеры описывают, как должны быть сериализованы атрибуты и отношения.

## Examples

### Использование сериализатора

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

Прочитайте Сериализаторы активной модели онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/9000/сериализаторы-активной-модели>

---

# глава 70: Соглашения об именах

## Examples

### Контроллеры

Названия классов контроллеров являются множественными. Причина в том, что контроллер управляет несколькими экземплярами экземпляра объекта.

*Например*: `OrdersController` будет контроллером для таблицы `orders`. Затем Rails ищет определение класса в файле с именем `orders_controller.rb` в `orders_controller.rb` `/app/controllers`.

*Например*: `PostsController` будет контроллером для таблицы `posts`.

Если имя класса контроллера имеет несколько заглавных букв, предполагается, что имя таблицы имеет знак подчеркивания между этими словами.

*Например*: если контроллер называется `PendingOrdersController` тогда предполагаемое имя файла для этого контроллера будет `pending_orders_controller.rb`.

### модели

Модель названа с использованием соглашения об именах классов для неразрывной `MixedCase` и всегда единственного имени таблицы.

*Например*: если таблица была названа `orders`, соответствующая модель была бы названа `Order`.

*Например*: если стол был назван `posts`, соответствующая модель будет называться `Post`.

Затем Rails ищет определение класса в файле `order.rb` в `order.rb` `/app/models`.

Если имя класса модели имеет несколько заглавных слов, предполагается, что имя таблицы имеет знак подчеркивания между этими словами.

*Например*: если модель называется `BlogPost` то предполагаемое имя таблицы будет `blog_posts`.

### Просмотры и макеты

Когда выполняется действие контроллера, Rails попытается найти соответствующий макет и представление на основе имени контроллера.

Представления и макеты размещаются в каталоге `app/views`.

Учитывая запрос к `PeopleController#index` , Rails будет искать:

- макет, называемый `people` в `app/views/layouts/` (или `application` если совпадение не найдено)
- представление под названием `index.html.erb` в `app/views/people/` по умолчанию
- если вы хотите отобразить другой файл с именем `index_new.html.erb` вам нужно написать код для этого в `PeopleController#index` action, например `render 'index_new'`
- мы можем установить разные layouts для каждого action , написав `render 'index_new', layout: 'your_layout_name'`

## Имена файлов и автозагрузка

Файлы Rails - и файлы Ruby в целом - должны быть названы с `lower_snake_case` файлов `lower_snake_case` . Например

```
app/controllers/application_controller.rb
```

это файл, содержащий определение класса `ApplicationController` . Обратите внимание, что, хотя `PascalCase` используется для имен классов и модулей, файлы, в которых они находятся, все равно должны быть `lower_snake_case` .

Согласованное именование важно, поскольку Rails использует файлы автоматической загрузки по мере необходимости и использует «перегиб» для преобразования между различными стилями именования, такими как преобразование `application_controller` в `ApplicationController` и обратно.

Например, если Rails видит, что класс `BlogPost` не существует (еще не загружен), он будет искать файл с именем `blog_post.rb` и попытаться загрузить этот файл.

Поэтому также важно назвать файлы для того, что они содержат, поскольку автозагрузчик ожидает, что имена файлов будут соответствовать содержимому. Если, например, `blog_post.rb` вместо этого содержит класс с именем `just Post` , вы увидите `LoadError: Expected [some path]/blog_post.rb to define BlogPost` .

Если вы добавите каталог под `app/something/` (например, `/models/` `/products/` ) и

- хотите, чтобы пространства имен включали модули и классы внутри нового каталога, вам не нужно ничего делать, и он будет загружен сам. Например, в `app/models/products/` you would need to wrap your class in модуле `Products`` .
- не хотите, чтобы модули и классы имен занимали пространство внутри моего нового `config.autoload_paths += %W( #{config.root}/app/models/products )` тогда вам нужно добавить `config.autoload_paths += %W( #{config.root}/app/models/products )` на ваш `application.rb` для автозагрузки.

Еще одна вещь, на которую нужно обратить внимание (особенно если английский не является вашим первым языком) является тот факт, что Rails учитывает нерегулярные

множественные существительные на английском языке. Поэтому, если у вас есть модель под названием «Foot», соответствующий контроллер нужно называть «FeetController», а не «FootController», если вы хотите использовать «магическую» маршрутизацию рельсов (и многие другие такие функции).

## Класс моделей из имени контроллера

Вы можете получить класс Model из имени контроллера таким образом (контекст - это класс Controller):

```
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
```

Прочитайте [Соглашения об именах онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/1493/соглашения-об-именах): <https://riptutorial.com/ru/ruby-on-rails/topic/1493/соглашения-об-именах>

---

# глава 71: Тестирование Rails-приложений

## Examples

### Модульный тест

Единичные тесты тестируют части приложения изолированно. обычно тестируемая единица является классом или модулем.

```
let(:gift) { create :gift }

describe '#find' do
  subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

#### ИСТОЧНИК

Этот вид, если тест является как можно более прямым и конкретным.

### Запрос теста

Запросные тесты - это сквозные тесты, которые имитируют поведение пользователя.

```
it 'allows the user to set their preferences' do
  check 'Ruby'
  click_on 'Save and Continue'
  expect(user.languages).to eq ['Ruby']
end
```

#### ИСТОЧНИК

Этот тип теста фокусируется на потоках пользователей и проходит через все уровни системы, иногда даже при обработке javascript.

Прочитайте Тестирование Rails-приложений онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/7853/тестирование-rails-приложений>

---

# глава 72: Фабричная девушка

## Examples

### Определение заводов

Если у вас есть класс ActiveRecord User с атрибутами имени и электронной почты, вы можете создать для него завод, сделав FactoryGirl угаданием:

```
FactoryGirl.define do
  factory :user do # it will guess the User class
    name      "John"
    email     "john@example.com"
  end
end
```

Или вы можете сделать это явным и даже изменить его имя:

```
FactoryGirl.define do
  factory :user_jack, class: User do
    name      "Jack"
    email     "jack@example.com"
  end
end
```

Затем в вашей спецификации вы можете использовать методы FactoryGirl с ними, например:

```
# 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)
```

Наиболее распространенными методами являются:

```
# 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)
```

Прочитайте [Фабричная девушка онлайн: https://riptutorial.com/ru/ruby-on-rails/topic/8330/фабричная-девушка](https://riptutorial.com/ru/ruby-on-rails/topic/8330/фабричная-девушка)

# глава 73: Форма Помощники

## Вступление

Rails предоставляет помощники вида для создания разметки формы.

## замечания

- Типы ввода даты , включая `date` , `datetime` , `datetime-local` , `time` , `month` И `week` не работают в FireFox.
- `input<type="telephone">` работает только с Safari 8.
- `input<type="email">` не работает в Safari

## Examples

### Создать форму

Вы можете создать форму с `form_tag` помощника `form_tag`

```
<%= form_tag do %>
  Form contents
<% end %>
```

Это создает следующий HTML-код

```
<form accept-charset="UTF-8" action="/" method="post">
  <input name="utf8" type="hidden" value="&#x2713;" />
  <input name="authenticity_token" type="hidden"
value="J7CBxfHalt49OSHp27hblqK20c9PgwJ108nDHX/8Cts=" />
  Form contents
</form>
```

Этот тег формы создал `hidden` поле ввода. Это необходимо, потому что формы не могут быть успешно представлены без него.

Второе поле ввода с именем `authenticity_token` добавляет защиту от `cross-site request forgery` .

### Создание формы поиска

Чтобы создать форму поиска, введите следующий код

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
```



```
<%= submit_tag("Search") %>
<% end %>
```

- `form_tag` : Это помощник по умолчанию для создания формы. Это первый параметр, `/search` - действие, а второй параметр - метод HTTP. Для форм поиска важно всегда использовать метод `get`
- `label_tag` : Этот помощник создает `<label> html </label>` .
- `text_field_tag` : это создаст элемент ввода с `text` типа
- `submit_tag` : создает элемент ввода с типом `submit`

## Помощники для элементов формы

### Флажки

```
<%= 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") %>
```

Это создаст следующий 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">I own a cat</label>
```

### Радио-кнопки

```
<%= 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") %>
```

Это генерирует следующий HTML-код

```
<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>
```

### Текстовая область

Чтобы создать более крупное текстовое поле, рекомендуется использовать `text_area_tag`

```
<%= text_area_tag(:message, "This is a longer text field", size: "25x6") %>
```

Это создаст следующий HTML-код

```
<textarea id="message" name="message" cols="25" rows="6">This is a longer text field</textarea>
```

## Числовое поле

Это создаст элемент `input<type="number">`

```
<%= number_field :product, :rating %>
```

Чтобы указать диапазон значений, мы можем использовать опцию `in`:

```
<%= number_field :product, :rating, in: 1..10 %>
```

## Поле пароля

Иногда вы хотите, чтобы символы, введенные пользователем, были замаскированы. Это создаст `<input type="password">`

```
<%= password_field_tag(:password) %>
```

## Поле электронной почты

Это создаст `<input type="email">`

```
<%= email_field(:user, :email) %>
```

## Телефонное поле

Это создаст `<input type="tel">`.

```
<%= telephone_field :user, :phone %>
```

## Помощники по дате

- `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) %>
```

## Падать

Стандартный пример: `@models = Model.all select_tag "models", options_from_collection_for_select (@models, "id", "name"), {}`

Это создаст следующий HTML-код: David

Последним аргументом являются опции, которые принимают следующее: `{multiple: false, disabled: false, include_blank: false, prompt: false}`

Дополнительные примеры можно найти:

[http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select\\_tag](http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag)

Прочитайте Форма Помощники онлайн: <https://riptutorial.com/ru/ruby-on-rails/topic/4509/форма-помощники>

---

# глава 74: Функция оплаты в рельсах

## Вступление

В этом документе вы можете представить вам, с полным примером, как вы можете реализовать различные способы оплаты с Ruby on Rails.

В этом примере мы рассмотрим две хорошо известные платежные платформы Stripe и Braintree.

## замечания

Документация.

[нашивка](#)

[Braintree](#)

## Examples

### Как интегрироваться с полосой

Добавить Stripe gem в наш Gemfile

```
gem 'stripe'
```

Добавьте файл `initializers/stripe.rb`. Этот файл содержит необходимые ключи для подключения к вашей учетной записи stripe.

```
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]
```

---

## Как создать нового клиента для Stripe

```
Stripe::Customer.create({email: email, source: payment_token})
```

Этот код создает нового клиента на Stripe с заданным адресом электронной почты и

ИСТОЧНИКОМ.

`payment_token` - это токен, указанный на стороне клиента, который содержит способ оплаты, например, кредитную карту или банковский счет. Дополнительная информация: [Stripe.js на стороне клиента](#)

---

## Как получить план из Stripe

```
Stripe::Plan.retrieve(stripe_plan_id)
```

Этот код извлекает план из Stripe по его идентификатору.

---

## Как создать подписку

Когда у нас есть клиент и план, мы можем создать новую подписку на Stripe.

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

Он создаст новую подписку и будет взимать плату с нашего Пользователя. Важно знать, что на самом деле происходит на Stripe, когда мы подписываемся на план пользователя, вы найдете здесь дополнительную информацию: [жизненный цикл Stripe Subscription](#).

---

## Как взимать плату с одного платежа

Иногда мы хотим заряжать наших пользователей всего один раз, потому что мы сделаем это дальше.

```
Stripe::Charge.create(amount: amount, customer: customer, currency: currency)
```

В этом случае мы взимаем с нашего пользователя один раз за заданную сумму.

Общие ошибки:

- Сумма должна быть отправлена в целочисленной форме, то есть 2000 будет составлять 20 единиц валюты. [Проверьте этот пример](#)
- Вы не можете взимать плату с пользователя в двух валютах. Если пользователь в последний момент взимал плату в евро, вы не можете взимать плату с пользователя в долларах США.
- Вы не можете заряжать пользователя без источника (способ оплаты).

Прочитайте [Функция оплаты в рельсах онлайн](https://riptutorial.com/ru/ruby-on-rails/topic/10929/функция-оплаты-в-рельсах): <https://riptutorial.com/ru/ruby-on-rails/topic/10929/функция-оплаты-в-рельсах>

# кредиты

S. No	Главы	Contributors
1	Начало работы с Ruby on Rails	<a href="#">Abhishek Jain</a> , <a href="#">Adam Lassek</a> , <a href="#">Ajay Barot</a> , <a href="#">animuson</a> , <a href="#">ArtOfCode</a> , <a href="#">Aswathy</a> , <a href="#">Community</a> , <a href="#">Darpan Chhatravala</a> , <a href="#">Darshan Patel</a> , <a href="#">Deepak Mahakale</a> , <a href="#">fybw id</a> , <a href="#">Geoffroy</a> , <a href="#">hschin</a> , <a href="#">hvenables</a> , <a href="#">Jon Wood</a> , <a href="#">kfrz</a> , <a href="#">Kirti Thorat</a> , <a href="#">Lorenzo Baracchi</a> , <a href="#">Luka Kerr</a> , <a href="#">MauroPorrasP</a> , <a href="#">michaelpri</a> , <a href="#">nifCody</a> , <a href="#">Niyanta</a> , <a href="#">olive_tree</a> , <a href="#">RADan</a> , <a href="#">RareFever</a> , <a href="#">Richard Hamilton</a> , <a href="#">sa77</a> , <a href="#">saadlulu</a> , <a href="#">sahil</a> , <a href="#">Sathishkumar Jayaraj</a> , <a href="#">Simone Carletti</a> , <a href="#">Stanislav Valášek</a> , <a href="#">theoretisch</a> , <a href="#">tpei</a> , <a href="#">Undo</a> , <a href="#">uzaif</a> , <a href="#">Yana</a>
2	ActionCable	<a href="#">Ich</a> , <a href="#">Sladey</a> , <a href="#">Undo</a>
3	ActionController	<a href="#">Adam Lassek</a> , <a href="#">Atul Khanduri</a> , <a href="#">Deep</a> , <a href="#">Fire-Dragon-DoL</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">jackerman09</a> , <a href="#">RamenChef</a> , <a href="#">Sven Reuter</a>
4	ActionMailer	<a href="#">Adam Lassek</a> , <a href="#">Atul Khanduri</a> , <a href="#">jackerman09</a> , <a href="#">owahab</a> , <a href="#">Phil Ross</a> , <a href="#">Richard Hamilton</a> , <a href="#">Rodrigo Argumedo</a> , <a href="#">William Romero</a>
5	ActiveJob	<a href="#">Brian</a> , <a href="#">owahab</a>
6	ActiveModel	<a href="#">Adam Lassek</a> , <a href="#">RamenChef</a>
7	ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">AnoE</a> , <a href="#">Bijal Gajjar</a> , <a href="#">br3nt</a> , <a href="#">D-side</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">glapworth</a> , <a href="#">jeffdill2</a> , <a href="#">Joel Drapper</a> , <a href="#">Luka Kerr</a> , <a href="#">maartenvanvliet</a> , <a href="#">marcamillion</a> , <a href="#">Mario Uher</a> , <a href="#">powerup7</a> , <a href="#">Sebastialonso</a> , <a href="#">Simone Carletti</a> , <a href="#">Sven Reuter</a> , <a href="#">walid</a>
8	ActiveSupport	<a href="#">Adam Lassek</a>
9	API Rails	<a href="#">Adam Lassek</a> , <a href="#">hschin</a>
10	Elasticsearch	<a href="#">Don Giovanni</a> , <a href="#">Luc Boissaye</a>
11	I18n - Интернационализация	<a href="#">Cyril Duchon-Doris</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">Frederik Spang</a> , <a href="#">gwcodes</a> , <a href="#">Jorge Najera T</a> , <a href="#">Lahiru</a> , <a href="#">RamenChef</a>
12	Mongoid	<a href="#">Ryan K</a> , <a href="#">tes</a>
13	Rails Cookbook - расширенные рецепты рельсов / методы обучения и	<a href="#">Milind</a>

	кодирования	
14	Rails Engine - модульные рельсы	<a href="#">Mayur Shah</a>
15	Rails -Engines	<a href="#">Deepak Kabbur</a>
16	Rails logger	<a href="#">Alejandro Montilla</a> , <a href="#">hgsongra</a>
17	RSpec и Ruby on Rails	<a href="#">Ashish Bista</a> , <a href="#">Scott Matthewman</a> , <a href="#">Simone Carletti</a>
18	Turbolinks	<a href="#">Mark</a>
19	Авторизация Rails 5 API	<a href="#">HParker</a>
20	Авторизация с помощью CanCan	<a href="#">4444</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">dgilperez</a> , <a href="#">mlabarca</a> , <a href="#">toobulkeh</a>
21	Активные вакансии	<a href="#">tirdadc</a>
22	Активные транзакции ActiveRecord	<a href="#">abhas</a> , <a href="#">Adam Lassek</a>
23	Ассоциации ActiveRecord	<a href="#">giniouxe</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">Khanh Pham</a> , <a href="#">Luka Kerr</a> , <a href="#">Manish Agarwal</a> , <a href="#">Niyanta</a> , <a href="#">RareFever</a> , <a href="#">Raynor Kuang</a> , <a href="#">Sapna Jindal</a>
24	Аутентификация API с помощью Devise	<a href="#">Vishal Taj PM</a>
25	Аутентификация пользователей в Rails	<a href="#">Abhinay</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">Antarr Byrd</a> , <a href="#">ArtOfCode</a> , <a href="#">dgilperez</a> , <a href="#">Kieran Andrews</a> , <a href="#">Luka Kerr</a> , <a href="#">Qchmq5</a> , <a href="#">uzaif</a> ,
26	Безопасная константа	<a href="#">Eric Bouchut</a> , <a href="#">Ryan K</a>
27	Безопасное сохранение ключей аутентификации	<a href="#">DawnPaladin</a>
28	Блокировка ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">fatfrog</a> , <a href="#">Muaaz Rafi</a>
29	Вложенная форма в Ruby on Rails	<a href="#">Arslan Ali</a>
30	Декоратор	<a href="#">Adam Lassek</a>
31	Добавить панель администратора	<a href="#">Ahsan Mahmood</a> , <a href="#">MSathieu</a>



32	Добавление RDS Amazon к вашему рельсу	<a href="#">Sathishkumar Jayaraj</a>
33	Драгоценные камни	<a href="#">Deep</a> , <a href="#">hschin</a> , <a href="#">ma_il</a> , <a href="#">MMachinegun</a> , <a href="#">RamenChef</a>
34	Загрузка файлов	<a href="#">Sergey Khmelevskoy</a>
35	Зарезервированные слова	<a href="#">Emre Kurt</a>
36	Идентификатор	<a href="#">Thang Le Sy</a>
37	Изменение стандартного приложения Rails	<a href="#">Whitecat</a>
38	Изменение часового пояса по умолчанию	<a href="#">Mihai-Andrei Dinculescu</a>
39	Импортировать все CSV-файлы из определенной папки	<a href="#">fool</a>
40	Инструменты для оптимизации и очистки кода Ruby on Rails	<a href="#">Akshay Borade</a>
41	Интеграция React.js с рельсами с использованием Hyperloop	<a href="#">Mitch VanDuyn</a>
42	Интерфейс запросов ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Ajay Barot</a> , <a href="#">Avdept</a> , <a href="#">br3nt</a> , <a href="#">dnsh</a> , <a href="#">Fabio Ros</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">ginioux</a> , <a href="#">jeffdill2</a> , <a href="#">MikeAndr</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">Niyanta</a> , <a href="#">powerup7</a> , <a href="#">rdnewman</a> , <a href="#">Reboot</a> , <a href="#">Robin</a> , <a href="#">sa77</a> , <a href="#">Vishal Taj PM</a>
43	Использование GoogleMaps с Rails	<a href="#">fiedl</a>
44	Команды генерации Rails	<a href="#">Adam Lassek</a> , <a href="#">ann</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">jackerman09</a> , <a href="#">Jeremy Green</a> , <a href="#">marcamillion</a> , <a href="#">Milind</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">nomatteus</a> , <a href="#">powerup7</a> , <a href="#">Reub</a> , <a href="#">Richard Hamilton</a>

45	Консолидация активов	<a href="#">fybw id</a> , <a href="#">Robin</a>
46	конфигурация	<a href="#">Ali MasudianPour</a> , <a href="#">Undo</a>
47	Креветка PDF	<a href="#">Awais Shafqat</a>
48	Кэширование	<a href="#">ArtOfCode</a> , <a href="#">Cuisine Hacker</a> , <a href="#">Khanh Pham</a> , <a href="#">RamenChef</a> , <a href="#">tirdadc</a>
49	маршрутизация	<a href="#">Adam Lassek</a> , <a href="#">advishnuprasad</a> , <a href="#">Ahsan Mahmood</a> , <a href="#">Alejandro Babio</a> , <a href="#">Andy Gauge</a> , <a href="#">AppleDash</a> , <a href="#">ArtOfCode</a> , <a href="#">Baldrick</a> , <a href="#">cl3m</a> , <a href="#">Cyril Duchon-Doris</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">Eliot Sykes</a> , <a href="#">esthervillars</a> , <a href="#">Fabio Ros</a> , <a href="#">Fire-Dragon-DoL</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">ginioux</a> , <a href="#">Giuseppe</a> , <a href="#">Hassan Akram</a> , <a href="#">Hizqeel</a> , <a href="#">HungryCoder</a> , <a href="#">jkdev</a> , <a href="#">John Slegers</a> , <a href="#">Jon Wood</a> , <a href="#">Kevin Sylvestre</a> , <a href="#">Kieran Andrews</a> , <a href="#">Kirti Thorat</a> , <a href="#">KULKING</a> , <a href="#">Leito</a> , <a href="#">Mario Uher</a> , <a href="#">Miliind</a> , <a href="#">Muhammad Faisal Iqbal</a> , <a href="#">niklashultstrom</a> , <a href="#">nuclearpidgeon</a> , <a href="#">pastullo</a> , <a href="#">Rahul Singh</a> , <a href="#">rap-2-h</a> , <a href="#">Raynor Kuang</a> , <a href="#">Richard Hamilton</a> , <a href="#">Robin</a> , <a href="#">rogerdpack</a> , <a href="#">Rory O'Kane</a> , <a href="#">Ryan Hilbert</a> , <a href="#">Ryan K</a> , <a href="#">Silviu Simeria</a> , <a href="#">Simone Carletti</a> , <a href="#">sohail khalil</a> , <a href="#">Stephen Leppik</a> , <a href="#">TheChamp</a> , <a href="#">Thor odinson</a> , <a href="#">Undo</a> , <a href="#">Zoran</a> ,
50	Миграции ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Aigars Cibulsksis</a> , <a href="#">Alex Kitchens</a> , <a href="#">buren</a> , <a href="#">Deepak Mahakale</a> , <a href="#">Dharam</a> , <a href="#">DSimon</a> , <a href="#">Francesco Lupo Renzi</a> , <a href="#">ginioux</a> , <a href="#">Hardik Kanjariya</a> <sup>٧</sup> , <a href="#">hschin</a> , <a href="#">jeffdill2</a> , <a href="#">Kirti Thorat</a> , <a href="#">KULKING</a> , <a href="#">maartenvanvliet</a> , <a href="#">Manish Agarwal</a> , <a href="#">Milo P</a> , <a href="#">Mohamad</a> , <a href="#">MZaragoza</a> , <a href="#">nomatteus</a> , <a href="#">Reboot</a> , <a href="#">Richard Hamilton</a> , <a href="#">rii</a> , <a href="#">Robin</a> , <a href="#">Rodrigo Argumedo</a> , <a href="#">rony36</a> , <a href="#">Rory O'Kane</a> , <a href="#">tessi</a> , <a href="#">uzaif</a> , <a href="#">webster</a>
51	Многоцелевые столбцы ActiveRecord	<a href="#">Fabio Ros</a>
52	Модельные состояния: AASM	<a href="#">Lomefin</a>
53	Наследование отдельных таблиц	<a href="#">Niyanta</a> , <a href="#">Ruslan</a> , <a href="#">Slava.K</a> , <a href="#">toobulkeh</a> , <a href="#">Vishal Taj PM</a>
54	Настройка углового с рельсами	<a href="#">B8vrede</a> , <a href="#">Rory O'Kane</a> , <a href="#">Umar Khan</a>
55	Неверная маршрутизация	<a href="#">Darpan Chhatravala</a>
56	Обновление рельсов	<a href="#">hschin</a> , <a href="#">michaelpri</a> , <a href="#">Rodrigo Argumedo</a>

57	Организация классов	<a href="#">Deep</a> , <a href="#">hadees</a> , <a href="#">HParker</a>
58	отладка	<a href="#">Adam Lassek</a> , <a href="#">Dénes Papp</a> , <a href="#">Dharam</a> , <a href="#">Kelseydh</a> , <a href="#">sa77</a> , <a href="#">titan</a>
59	Проверка ActiveRecord	<a href="#">Adam Lassek</a> , <a href="#">Colin Herzog</a> , <a href="#">Deepak Mahakale</a> , <a href="#">dgilperez</a> , <a href="#">dodo121</a> , <a href="#">giniouxe</a> , <a href="#">Hai Pandu</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">mmichael</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">pablofullana</a> , <a href="#">Richard Hamilton</a>
60	Просмотры	<a href="#">danirod</a> , <a href="#">dgilperez</a> , <a href="#">elasticman</a> , <a href="#">Luka Kerr</a> , <a href="#">MikeC</a> , <a href="#">MMachinegun</a> , <a href="#">Pragash</a> , <a href="#">RareFever</a>
61	Развертывание приложения Rails на Heroku	<a href="#">B Liu</a> , <a href="#">hschin</a>
62	Рамки Rails на протяжении многих лет	<a href="#">Shivasubramanian A</a>
63	Реагировать с рельсами, используя камень с ребрами	<a href="#">Kimmo Hintikka</a> , <a href="#">tirdadc</a>
64	Рекомендации по Rails	<a href="#">Adam Lassek</a> , <a href="#">Brandon Williams</a> , <a href="#">Gaston</a> , <a href="#">giniouxe</a> , <a href="#">Hardik Upadhyay</a> , <a href="#">inye</a> , <a href="#">Joel Drapper</a> , <a href="#">Josh Caswell</a> , <a href="#">Luka Kerr</a> , <a href="#">ma_il</a> , <a href="#">msohng</a> , <a href="#">Muaaz Rafi</a> , <a href="#">piton4eg</a> , <a href="#">powerup7</a> , <a href="#">rony36</a> , <a href="#">Sri</a> , <a href="#">Tom Lazar</a>
65	Рельсы 5	<a href="#">thiago araujo</a>
66	Рельсы на докере	<a href="#">ppascualv</a> , <a href="#">Sathishkumar Jayaraj</a>
67	Сериализаторы активной модели	<a href="#">Flip</a> , <a href="#">owahab</a>
68	Соглашения об именах	<a href="#">Andrey Deineko</a> , <a href="#">Atul Khanduri</a> , <a href="#">br3nt</a> , <a href="#">Flambino</a> , <a href="#">giniouxe</a> , <a href="#">hgsongra</a> , <a href="#">Luka Kerr</a> , <a href="#">Marko Kacanski</a> , <a href="#">Muhammad Abdullah</a> , <a href="#">Sven Reuter</a> , <a href="#">Xinyang Li</a>
69	Тестирование Rails-приложений	<a href="#">HParker</a>
70	Фабричная девушка	<a href="#">Rafael Costa</a>
71	Форма Помощники	<a href="#">aisflat439</a> , <a href="#">owahab</a> , <a href="#">Richard Hamilton</a> , <a href="#">Simon Tsang</a> , <a href="#">Slava.K</a>
72	Функция оплаты в рельсах	<a href="#">ppascualv</a> , <a href="#">Sathishkumar Jayaraj</a>