

 無料電子ブック

学習

Ruby on Rails

Free unaffiliated eBook created from
Stack Overflow contributors.

#ruby-on-
rails

.....	1
1: Ruby on Rails	2
.....	2
.....	2
Examples.....	3
Ruby on Rails.....	3
RSpecRails.....	5
.....	6
.....	7
Rails.....	7
JSONRails API.....	8
Rails.....	9
2: ActionCable	12
.....	12
Examples.....	12
[].....	12
[]Coffeescript.....	12
app / assets / javascripts / channels / notifications.coffee	12
app / assets / javascripts / application.js	12
app / assets / javascripts / cable.js	13
.....	13
3: ActionController	14
.....	14
Examples.....	14
HTMLJSON.....	14
.....	15
.....	15
.....	15
.....	16
.....	16
404.....	18

REST	18
.....	19
.....	20
.....	22
RescueActiveRecord :: RecordNotFound with redirect_to	24
4: ActionMailer	25
.....	25
.....	25
Examples	25
.....	25
user_mailer.rb	25
user.rb	25
approved.html.erb	26
approved.text.erb	26
.....	26
.....	26
ActionMailer	27
.....	27
ActionMailer	34
5: ActiveRecord	36
.....	36
Examples	36
ActiveRecord :: Validations	36
6: ActionController	37
Examples	37
.....	37
.....	37
.....	38
/	38
.....	38
.....	39
.....	

.....	39
.....	41
.....	41
.....	41
7: ActiveRecord	43
.....	43
Examples.....	43
.....	43
1 ActiveRecord.....	43
.....	43
.....	44
.....	44
.....	44
8: ActiveRecord	46
.....	46
Examples.....	46
.....	46
9: ActiveRecord	47
Examples.....	47
.....	47
.....	47
10: ActiveRecord	48
.....	48
.....	48
Examples.....	48
.....	48
.....	49
.....	49
.....	50
.....	50
.....	50

.....	51
.....	51
.....	52
.....	52
.....	53
3.....	53
.....	53
.....	53
.....	54
.....	54
.....	54
.....	54
.....	55
.....	55
.....	55
.....	56
.....	56
hstore.....	56
.....	57
.....	57
NOT NULL.....	57
11: ActiveRecord.....	59
Examples.....	59
.....	59
.....	59
.....	60
.....	60
.....	61
.....	61
.....	61
ActiveModel::Validatorvalidates_with.....	61
ActiveModel::EachValidatorvalidate.....	62
.....	62
.....	63
.....	63

.....	63
on.....	64
12: ActiveRecord.....	65
.....	65
Examples.....	65
.where.....	65
.....	66
.....	66
.....	67
.....	67
ActiveRecord Bang.....	68
.find_by.....	68
.....	69
ActiveRecord.....	69
.....	69
.group.count.....	70
.distinct.uniq.....	71
.....	71
.....	72
.....	72
13: ActiveSupport.....	74
.....	74
Examples.....	74
.....	74
Stringat.....	74
Stringfrom.....	74
String to.....	74
.....	75
.....	75
/.....	75
to_time.....	75
to_date.....	75

to_datetime	76
.....	76
.....	76
.....	76
squish	76
remove	76
.....	76
truncate_words	77
strip_heredoc	77
String Inflection	78
.....	78
singularize	78
constantize	78
Stringsafe_constantize	78
Stringcamelize	79
titleize	79
.....	79
#dasherize	79
demodulize	80
deconstantize	80
Stringparameterize	80
tableize	80
Stringclassify	81
humanize	81
upcase_first	81
foreign_key	81
14: CanCan	82
.....	82
.....	82
Examples	82

CanCan.....	82
.....	83
.....	83
.....	85
15: DeviseApi.....	86
.....	86
Examples.....	86
.....	86
.....	86
16: GoogleMapsRails.....	88
Examples.....	88
Googlejavascript.....	88
.....	89
Google.....	89
javascript.....	90
.....	91
.....	92
.....	93
json.....	93
.....	93
.....	94
.....	94
17: HerokuRails.....	96
Examples.....	96
.....	96
Heroku.....	99
18: I18n -	101
.....	101
Examples.....	101
I18n.....	101
I18n.....	101
.....	102

.....	102
URL	103
.....	103
.....	104
HTTP	104
.....	105
1.....	105
2. CloudFlare.....	105
ActiveRecord.....	105
HTML18n.....	107
19: Prawn PDF	109
Examples.....	109
.....	109
.....	110
.....	110
.....	110
.....	110
20: Rails 5	111
Examples.....	111
Ruby on Rails 5 API.....	111
Ruby on Rails 5RVM.....	113
21: Rails 5 API	114
Examples.....	114
Railsauthenticate_with_http_token.....	114
22: Rails API	115
Examples.....	115
API.....	115
23: Rails Cookbook - /	116
Examples.....	116
.....	116
Rails -	117

- `where 'for`	117
24: railsAmazon RDS	118
.....	118
Examples	118
MYSQL RDS	118
25: Rails	120
Examples	120
.....	120
.....	120
26: Rails -	121
.....	121
.....	121
Examples	121
.....	121
Todo	122
27: Rails	124
.....	124
.....	124
Examples	124
Devise	124
.....	125
.....	125
.....	125
has_secure_password	126
.....	126
has_secure_password	126
has_secure_token	126
28: Rails	128
Examples	128
101	128
1Rails	128

2.....	128
3AngularJS.....	128
4.....	129
5.....	129
29: Rails.....	131
Examples.....	131
Rails 4.2Rails 5.0.....	131
30: Rails.....	133
.....	133
.....	133
.....	133
Examples.....	134
Rails.....	134
Rails.....	134
.....	135
.....	136
31: Rails.....	137
Examples.....	137
.....	137
.....	137
.....	138
default_scope.....	139
default_scopeorder.....	139
default_scope.....	139
unscoped.....	139
unscoped.....	140
default_scope.....	140
YAGNI.....	141
.....	141
Overengineering.....	141
.....	141
.....	141

.....	141
.....	142
-	142
YAGNI -	142
.....	142
.....	142
32: Rails	145
Examples	145
Rails.logger	145
33: RSpecRuby on Rails	146
.....	146
Examples	146
RSpec	146
34: Ruby on Rails	147
.....	147
Examples	147
.....	147
35: Ruby on Rails	148
Examples	148
Ruby on Rails	148
36:	150
.....	150
Examples	150
.....	150
.....	150
37:	151
Examples	151
.....	151
.....	151
.....	151

38:	152
	152
Examples	152
	152
39:	153
Examples	153
	153
has_one	153
	154
	154
has_manythrough	154
has_onethrough	155
has_and_belongs_to_many	155
	156
40:	157
Examples	157
	157
SQL	157
	158
	159
HTTP	159
	160
41:	161
	161
Examples	161
	161
	161
42:	165
	165
	165
	165
Examples	165

.....	165
.....	166
.....	166
.....	166
.....	167
.....	167
.....	167
43:	169
.....	169
Examples	169
SimpleDelegator	169
Draper	170
44:	171
Examples	171
Rails	171
IDE	171
Ruby on Rails+	173
Ruby / Rails	173
1. Exception.inspect.inspect	173
2. IRBbybugpry	173
.....	173
Ruby	174
pryruby-on-rails	175
45: Rails	177
.....	177
Examples	177
.....	177
.....	177
46:	178
.....	178
Examples	178

RailsActive RecordUTC.....	178
RailsActive Record.....	178
47:	180
.....	180
Examples.....	180
.....	180
48: React.jsRails	182
.....	182
.....	182
Examples.....	182
Rails.....	182
.....	183
HTML.....	183
.....	183
.....	183
.....	184
49:	185
Examples.....	185
.....	185
.....	185
.....	185
AssetTagHelper.....	186
.....	186
image_path.....	186
image_url.....	186
image_tag.....	186
JavaScript	186
javascript_include_tag.....	186
javascript_path.....	186
javascript_url.....	187
.....	187

stylesheet_link_tag.....	187
stylesheet_path.....	187
stylesheet_url.....	187
.....	187
.....	188
HTML.....	188
HAML -	188
50:	191
Examples.....	191
Carrierwave.....	191
-	191
51:	193
.....	193
.....	193
Examples.....	193
.....	193
.....	193
.....	194
.....	194
.....	194
.....	194
.....	195
.....	195
.....	195
.....	195
.....	195
.....	195
.....	196
52: ID	197
.....	197
Examples.....	197
Rails.....	197

Gemfile	197
app / models / user.rb	197
h11	197
h12	197
53: AASM	199
Examples.....	199
AASM.....	199
54:	201
Examples.....	201
.....	201
.....	201
.....	202
.....	202
.....	203
.....	203
55:	204
.....	204
.....	204
Examples.....	204
.....	204
.....	206
.....	208
.....	211
.....	211
.....	212
URL.....	213
.....	213
RESTful.....	213
.....	214
.....	214
.....	215
.....	215

.....	215
56: -	217
.....	217
.....	217
.....	217
.....	217
Examples.....	217
.....	217
57:	218
.....	218
.....	218
Examples.....	218
.....	218
.....	218
.....	218
.....	219
1	219
58:	220
.....	220
Examples.....	220
.....	220
59:	227
.....	227
Examples.....	227
.....	227
.....	228
STIRails.....	228
60:	229
Examples.....	229
rails_react gemRails.....	229
react_rails.....	229
.....	

61:	232
Examples	232
.....	232
.....	232
.....	232
.....	233
.....	233
62: ActiveRecord	235
.....	235
Examples	235
.....	235
.....	235
.....	235
.....	235
63:	236
Examples	236
.....	236
safe_constantize	236
64:	237
.....	237
Gemfile	237
Examples	237
.....	237
Rails	237
Gemfile	237
Gemfile.lock	237
.....	238
.....	238
Gemfiles	238
.....	239

65:	242
Examples	242
.....	242
66:	243
Examples	243
.....	243
.....	243
.....	244
Searchkick	244
67:	246
Examples	246
.....	246
68:	248
Examples	248
Rails	248
.....	248
Rails	249
.....	249
.....	250
69:	251
Examples	251
1	251
70: CSV	252
.....	252
Examples	252
CSV	252
71:	254
.....	254
.....	254
.....	254
Examples	254

rails_admin gem	254
72:	258
.....	258
Examples	258
.....	258
73:	260
.....	260
Examples	260
.....	260
.....	260
.....	261
74: Rails	262
.....	262
Examples	262
Rails	262
Rails 1.xRails	262
Rails 2.xRails	262
Rails 3.xRails	262
.....	264

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

1: Ruby on Rails スタートガイド



Ruby on Rails (RoR)、すなわち Rails は、オープンソースの Web アプリケーションフレームワークです。 Rails は、Ruby、HTML、CSS、および JavaScript をして、Web サーバーである Web アプリケーションをします。 Rails は MVC Model-View-Controller パターンをし、データベースからビューまでのライブラリをします。

バージョン

バージョン	
5.1.2	2017-06-26
5.0	2016630
4.2	20141219
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	2007127
1.2	20070119
1.1	2006-03-28
1.0	2005-12-13

Examples

Ruby on Rails アプリケーションの

ここでは、*Ruby*と*Ruby on Rails*がすでにしくインストールされていることをとしています。そうでない、[ここでそれをうをつけることができます](#)。

コマンドラインまたはターミナルをきます。しいrailsアプリケーションをするには、`rails new`コマンドのにアプリケーションのをけてします。

```
$ rails new my_app
```

のRailsバージョンでRailsアプリケーションをするは、アプリケーションのにできます。これをするには、`rails _version_ new`にアプリケーションのをけてします。

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

これはとばれるRailsアプリケーションをします `MyAppmy_app` ディレクトリを、すでににされているのをインストール `Gemfile` して `bundle install`。

しくしたアプリのディレクトリにりえるには、`cd` コマンドをします。これは、`change directory` をします。

```
$ cd my_app
```

`my_app` ディレクトリには、Railsアプリケーションのをするいくつかのファイルとフォルダがあります。は、デフォルトでされるファイルとフォルダのです。

ファイルフォルダ	
app /	アプリケーションのコントローラ、モデル、ビュー、ヘルパー、メーラ、およびアセットがまれます。
bin/	あなたのアプリケーションをするルールスクリプトをみ、アプリケーションの、デプロイ、またはにするのスクリプトをむことができます。
config /	アプリケーションのルート、データベースなどをします。
config.ru	アプリケーションをするためにされるラックベースのサーバのためのラック。
db /	のデータベーススキーマとデータベースがまれます。
Gemfile Gemfile.lock	これらのファイルをすると、Railsアプリケーションになgemのをできま

ファイルフォルダー	
	す。これらのファイルはBundler gemによってされます。
lib /	アプリケーションのモジュール。
ログ/	アプリケーションログファイル。
パブリック/	でられるのフォルダです。ファイルとコンパイルみのアセットがまれます。
Rakefile	このファイルは、コマンドラインからできるタスクを試してみます。タスクは、Railsのコンポーネントでされています。
README.md	これはアプリケーションのなです。このファイルをして、のあなたのアプリケーションがをしているのか、どのようにするのかなどを教えてください。
テスト/	ユニットテスト、フィクスチャ、およびそのテスト。
/	ファイルキャッシュやpidファイルなど。
ベンダー/	すべてのサードパーティコードの。なRailsアプリケーションでは、これはされたをみます。

これで、 database.yml ファイルからデータベースをすることがあります

5.0

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

5.0

```
rake db:create
```

データベースをしたので、テーブルをするためにマイグレーションをすることがあります。

5.0

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

5.0

```
rake db:migrate
```

アプリケーションをするには、サーバーをするがあります。

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

デフォルトでは、レールはポート3000でアプリケーションをします。なるポートでアプリケーションをするには、

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

ブラウザで[http:// localhost3000](http://localhost3000)にすると、アプリケーションがであることをすRailsのようこそページがされます。

エラーがすると、いくつかのがするがあります。

- `config/database.yml` **があり** `config/database.yml`
- インストールされていない `Gemfile` **があります。**
- **の** `rails db:migrate` `rails db:migrate`
- **の** `rails db:rollback` **する** `rails db:rollback`

それでもエラーがしたは、 `config/database.yml` してください。

したデータベースと **RSpec** テストツールをむしい **Rails** アプリケーションをする

Railsはデフォルトのデータベースとして `sqlite3` をしますが、あなたがんだデータベースをってしいrailsアプリケーションをすることができます。 `-d` オプションのにデータベースのをするだけです。

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

これは、なデータベースオプションのリストです

- `mysql`
- オラクル
- `postgresql`
- `sqlite3`
- フロントベース
- `ibm_db`
- SQL サーバー
- `jdbcmysql`
- `jdbcsqlite3`
- `jdbcpostgresql`
- `jdbc`

`-T` コマンドは、 `minitest` のインストールをスキップすることをします。 **RSpec** のようなのテスト

スイートをインストールするには、 [Gemfile](#)をして、

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

に、コンソールからのコマンドをします。

```
rails generate rspec:install
```

コントローラの

コントローラ例えば `Posts` をするには、コマンドラインまたはターミナルからプロジェクトディレクトリにし、 のコマンドをします。

```
$ rails generate controller Posts
```

このコードを `generate` は、 `generate` を `g` にきえ `generate`

```
$ rails g controller Posts
```

しくされた `app/controllers/posts_controller.rb` をくと、アクションのないコントローラがされます

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

コントローラーのをすことで、コントローラーのデフォルトのメソッドをすることができます。

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

モジュールにコントローラをするには、コントローラを `parent_module/controller_name` ようなパスでします。例えば

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

これにより、 のファイルがされます。

```
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
```

コントローラは、に `ApplicationController` からするようにされたクラスです。

このクラスのに、このコントローラーのアクションになるメソッドをします。

でリソースをする

guides.rubyonrails.orgから

モデルをするわりに。。。。をしましょう。 Railsのは、モデルのフルセット、そのモデルのデータベース、それをするコントローラ、データをおよびするためのビュー、およびのそれぞれのためのテストスイートです。

とテキストをつTaskというリソースをにしたをにします。

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

これにより、のファイルがされます。

```
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
```

Taskとばれるリソースのscaffoldによってされたファイルをする

```
rails destroy scaffold Task
```

のデータベースアダプタをしてしい**Rails**アプリケーションをする

Railsはデフォルトで、 **じ**のパターンからしたORMObject Relational MappingであるActiveRecordでされます。

ORMとして、リレーショナル・マッピングをするように、よりにはSQLリクエストをするようにされているため、SQLデータベースのみにがあります。

ただし、のデータベースシステムをしてRailsアプリケーションをすることはできます。

1. アクティブレコードなしでアプリをするだけです

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

- 2.

のデータベースシステムをGemfileする

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

3. `bundle install`し、のデータベースからインストールにいます。

このでは、`mongoid`のマッピングであるMongoDBレールのためにされたくののデータベースとして--それはまたから`ActiveModel`とじように`ActiveRecord`ような、コールバック、などのくののためのインタフェースをし、。

のデータベースアダプタにはのものがありませんが、これらにされません。

- データマップ
-

JSONでのRails APIの

このでは、Railsアプリケーションのがあることをとしています。

Rails 5にAPIアプリケーションをするには、

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

Gemfileでactive_model_serializersをする

```
gem 'active_model_serializers'
```

にバンドルをインストールする

```
bundle install
```

する`ActiveModelSerializer`アダプタをします:`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 )
```

あなたのリソースのためのしいをする

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

これにより、のファイルがされます。

コントローラ `app / controllers / tasks_controller.rb`

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

Railsのインストール

UbuntuにRailsをインストールする

きれいなubuntuで、Railsのインストールはまっすぐむべきです

ubuntuパッケージのアップグレード

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

RubyとRailsのをインストールする

```
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
```

Rubyバージョンマネージャのインストール。この、なものはrbenvをしています

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

Rubyビルドのインストール

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

シェルをする

```
exec $SHELL
```

ルビーをインストールする

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

レールのりけ

```
gem install rails
```

WindowsにRailsをインストールする

ステップ1 *Ruby*のインストール

Rubyプログラミングがインストールされている場合があります。RubyInstallerと呼ばれるRubyのプリコンパイルをすることができます。

- rubyinstaller.orgからRubyインストーラをダウンロードしてします。
- インストーラをします。"あなたのPATHにRubyファイルを"をチェックし、インストールしてください。
- Rubyにアクセスするには、Windowsのメニューからすべてのプログラムをクリックし、Rubyまでスクロールして、「Rubyでコマンドプロンプトを」をクリックします。コマンドプロンプトターミナルがきます。 `ruby -v`としてEnterキーをすと、インストールしたRubyのバージョンがされます。

ステップ2 *Ruby*キット

Rubyをインストールしたら、Railsをインストールしようとすることができます。しかし、いくつかのRailsライブラリは、コンパイルするためにいくつかのビルドツールをとしており、デフォルトではWindowsにはそれらのツールがありません。 Rails Gem::InstallError: The '[gem name]' native gem requires installed build tools. をインストールしようするとエラーがされたは、これをできます Gem::InstallError: The '[gem name]' native gem requires installed build tools. これをするには、Ruby Development Kitをインストールするがあります。

- DevKitをダウンロードする
- インストーラをします。
- DevKitをにインストールするフォルダをするがあります。はあなたのハードドライブのルート、 `C:\RubyDevKit`インストールすることをおめします。ディレクトリにはスペースをしないでください。

は、DevKitツールをRubyでできるようにするがあります。

- コマンドプロンプトで、DevKitディレクトリにします。 `cd C:\RubyDevKit`またはそれをインストールしたディレクトリ。
- DevKitのをするには、Rubyスクリプトをするがあります。 `ruby dk.rb init`し `ruby dk.rb install`。はRubyインストールにDevKitをするためのスクリプトをします。

DevKitは、しいライブラリをインストールするにRubyツールでできるようになりました。

ステップ3 *Rails*

これでRailsをインストールできます。 RailsはRubyのとしてします。コマンドプロンプトでのようにします。

```
gem install rails
```

Enterをすと、gemプログラムはRailsのバージョンをダウンロードしてインストールし、Railsがしているのすべてのものをインストールします。

ステップ4 **Node.js**

Railsがしているライブラリの中には、JavaScriptランタイムのインストールがなものがあります。それらのライブラリがしくするようにNode.jsをインストールしましょう。

- [ここからNode.jsインストーラをダウンロードしてください](#)。
- ダウンロードがしたら、ダウンロードフォルダにアクセスし、node-v4.4.7.pkgインストーラをしnode-v4.4.7.pkg。
- なをみ、にし、ウィザードのりのので「へ」をクリックし、すべてをデフォルトにします。
- あなたのコンピュータにをえることをするかどうかをねるウィンドウがポップアップすることがあります。[はい]をクリックします。
- インストールがしたら、RailsがNode.jsにアクセスできるようにコンピュータをすることがあります。

おいのコンピュータがしたら、Windowsのメニューから「すべてのプログラム」をクリックし、Rubyまでスクロールして、「Rubyでコマンドプロンプトを」をクリックすることをれないでください。

オンラインでRuby on Railsスタートガイドをむ <https://riptutorial.com/ja/ruby-on-rails/topic/225/ruby-on-rails>スタートガイド

2: ActionCable

ActionCableはRails 4.xででき、Rails 5にバンドルされています。サーバーとクライアントのリアルタイムにWebソケットをにできます。

Examples

□ サーバー

```
# 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
```

□ クライアントサイド Coffeescript

app / assets / javascripts / channels / notifications.coffee

```
App.notifications = App.cable.subscriptions.create "NotificationsChannel",
  connected: ->
    # Called when the subscription is ready for use on the server
    $(document).on "change", "input", (e)=>
      @notify(e.target.value)

  disconnected: ->
    # Called when the subscription has been terminated by the server
    $(document).off "change", "input"

  received: (data) ->
    # Called when there's incoming data on the websocket for this channel
    $('body').append(data)

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

app / assets / javascripts / application.js この

ようにされます

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

app / assets / javascripts / cable.js このように されます

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

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

  App.cable = ActionCable.createConsumer();

}).call(this);
```

ユーザ

```
# 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
```

オンラインでActionCableをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1498/actioncable>

3: ActionController

き

Action ControllerはMVCのCです。ルーターがに作るコントローラをした、コントローラはをしてをします。

コントローラはをけり、モデルからデータをフェッチまたはし、ビューをしてをします。コントローラは、モデルとビューののとえることができます。モデルデータをビューでできるようにしてユーザーにし、ユーザーデータをモデルにまたはします。

Examples

HTMLのわりにJSONをする

```
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
```

さらに、ルートがになります

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

これは/usersのリクエストにして2つのなるでし/users

- /usersまたは/users.htmlアクセスすると、Hello Worldのコンテンツをむhtmlページがされます
- /users.jsonにアクセスすると、をむJSONオブジェクトがされます。

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

あなたのルートがJSONリクエストだけにえるformat.html { render inline: "Hello World" }は、format.html { render inline: "Hello World" }をすることができます。

コントローラ

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

これはなコントローラであり、のルートがされています routes.rb。

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

URL `/users` アクセスすると、 `Hello World` メッセージが Web ページにされ `/users`

パラメーター

コントローラは HTTP パラメータにアクセスできます URL で `?name=foo` としてされるかもしれませんが、Ruby on Rails もさまざまなフォーマットをします。GET パラメータと POST パラメータをするはありませんが、どのようなでもそれをすべきではありません。

```
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
```

いつものようにたちのルート

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

URL `/users?name=john` アクセスします `/users?name=john` とは `Hello John`、 `access` `/users?name=whatever` とは `Hello someone` なります

フィルタリングパラメータ

```
class UsersController < ApplicationController
  def index
    respond_to do |format|
      format.html do
        render html: "Hello #{ user_params[:name] } user_params[:sentence]"
      end
    end
  end
end
```

```

    end
  end
end

private

def user_params
  if params[:name] == "john"
    params.permit(:name, :sentence)
  else
    params.permit(:name)
  end
end
end
end

```

いくつかのパラメータをまたはすることで、なものだけをさせることができ、するつもりはないユーザーオプションのようなきはありません。

`/users?name=john&sentence=developer` されます `Hello john developer` しかしれ、
`/users?name=smith&sentence=spy` する `Hello smith` ので、の `:sentence` あなたのようにアクセスするとき
にのみされている `john`

リダイレクト

ルートをします。

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

をしてのURLにリダイレクトすることができます。

```

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

```

ユーザーがしたのページにすることができます

```
redirect_to :back
```

*Rails 5*では、リダイレクトするためのがなることにしてください。

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

のページにリダイレクトしようとしてますが、でないブラウザはHTTP_REFERERヘッダーをブロックしています、`:fallback_location` リダイレクトされ、`:fallback_location`

ビューの

ルートをします。

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

コントローラー

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

ビュー `app/users/index.html.erb` がレンダリングされます。ビューがの

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

は「**Hello World**」というテキストのWebページになります。

のビューをレンダリングするは、のものをできます。

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

わりに `app/views/pages/home.html.erb` ファイルがわりにされます。

コントローラインスタンスをしてビューにをすことができます

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

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

そして、 `app/views/users/index.html.erb` ファイルで `@name` をうことができます

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

そしてはのようになります "こんにちはジョン "

レンダリングについてながありますが、 `render` をにすることができます。すると、 Railsはこれをします。そう

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

```
end
```

わりにのようによくことができます

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

Railsは、`app/views/users/index.html.erb` ファイルをレンダリングするがあることをするのにスマート `app/views/users/index.html.erb`。

レコードが見つからないは**404**

またはページをするのではなく、レコードからしてください。

```
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
```

な**REST**コントローラ

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]

  def index
    @posts = Post.all
  end

  def show

  end

  def new
    @post = Post.new
  end

  def edit

  end

  def create
    @post = Post.new(post_params)

    respond_to do |format|
      if @post.save
```

```

    format.html { redirect_to @post, notice: 'Post was successfully created.' }
    format.json { render :show, status: :created, location: @post }
  else
    format.html { render :new }
    format.json { render json: @post.errors, status: :unprocessable_entity }
  end
end
end

def update
  respond_to do |format|
    if @post.update(post_params)
      format.html { redirect_to @post.company, notice: 'Post was successfully updated.' }
      format.json { render :show, status: :ok, location: @post }
    else
      format.html { render :edit }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end

def destroy
  @post.destroy
  respond_to do |format|
    format.html { redirect_to posts_url, notice: '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

```

のエラーページをする

な「しありませんが、かがっていた」のではなく、のあるエラーをユーザにしたい、Railsはこのためのらしいユーティリティをとっています。

app/controllers/application_controller.rb ファイルをき、のようなコードをつけてください

```

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

```

のエラーからするために `rescue_from` をできるようになりました。

```

class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
  rescue_from ActiveRecord::RecordNotFound, with: :record_not_found
end

```

```
private

def record_not_found
  render html: "Record <strong>not found</strong>", status: 404
end
end
```

Exception または StandardError からしないことがされます。さもなければ、Rails はエラーのにつページをできません。

フィルタ

フィルタは、コントローラアクションの、またはにされるメソッドです。これらはされているため、 ApplicationController にのをすると、 ApplicationController がけるごとにされます。

フィルタ

フィルタがコントローラのにされ、をすおよび/またはリダイレクトするに、ないは、ユーザーがログインしているかどうかをすることです。

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

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

リクエストがコントローラアクションにするにフィルタがリクエストでされる。をし、アクションをにバイパスすることができます。

before フィルタのそののなは、リクエストをすするためにされたアクションへのアクセスをすに、ユーザーのをすることです。また、データベースからリソースをロードしたり、リソースのアクセスをチェックしたり、のでリダイレクトをしたりするのにもされています。

フィルターの

フィルターは「」とていますが、アクションのにされるので、しようとしているオブジェクトにアクセスできます。したがって、アクションがしたでフィルタがされたです。それはをすることができます。たいていの、のフィルタでかがわれると、アクションでできますが、のアクションをしたにするロジックがあるは、のフィルタがなですそれ。

に、はロギングにされたフィルターのをてきました。

フィルターのり

フィルターのりには、アクションがされるにがあるがあります。それは、なでアクションにびつくだけです。それはアクションにするはなく、のフィルターのようにするはありません。

フィルタのりは、ラックミドルウェアがどのようにするかとに、によってするをするがあります。

のコールバックはアクションのをラップします。りしコールバックを2つのなるスタイルでくことができます。は、コールバックはコードののチャンクです。そのコードは、アクションがされるにびされます。コールバックコードがyieldをびすと、アクションがされます。アクションがすると、コールバックコードはをしします。したがって、yieldののコードはbeforeアクションのコールバックのようになり、yieldののコードはafterアクションのコールバックになります。コールバックコードがyieldをびさないアクションはされません。これは、のアクションコールバックにfalseをすこととじです。

aroundフィルタのをにしします。

```
around_filter :catch_exceptions

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

これはのアクションのをキャッチし、メッセージをログにします。、セットアップとティアダウン、そのののにフィルタをすることができます。

のみと

すべてのフィルタは、をしてのアクションにできます :onlyおよび: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
```

フィルターをスキップする

すべてのフィルタされたフィルタもは、のアクションにしてスキップすることもできます。

```
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

```

されているので、フィルタは `namespace 「」` コントローラですることでもでき `namespace`。たとえば、`admin` があり、`admin` ユーザーだけがアクセスできるようにしたいとします。のようなことができます

```

# 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

```

Rails 4.xでは `before_filter` を `before_action` と `before_action` することができますが、`before_filter` は **Rails 5.0.0** であり、**5.1**でされることにしてください。

コントローラの

Railsはもちろん、コントローラーのく `ジェネレーター` をしています。

あなたのアプリケーションフォルダでこのコマンドをすると、しいコントローラをすることができます

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

`rails g` エイリアスをして、`rails generate` をびすこともでき `rails g`

たとえば、`#index` アクションと `#show` アクションを `#show` して `Product` モデルのコントローラをするには

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

これにより、したのアクションをつコントローラが `app/controllers/products_controller.rb` にされます

```
class ProductsController < ApplicationController
  def index
  end

  def show
  end
end
```

また、コントローラのアクションの2つのテンプレート `index.html.erb` と `show.html.erb` を `app/views/products` フォルダにします。はテンプレートエンジンによってなるがあるのでしてください。して「`slim`」えば、がされ、 `index.html.slim` と `show.html.slim`

さらに、アクションをしたは、 `routes` ファイルにもされます

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

Railsは `app/helpers/products_helper.rb` ヘルパーファイルをし、 `app/assets/javascripts/products.js` と `app/assets/stylesheets/products.css` のアセットファイルもし `app/assets/stylesheets/products.css` 。ビューについてはジェネレータが `Gemfile` されたものによってこのをします。つまり、アプリケーションで `Coffeescript` と `Sass` をしている、コントローラジェネレータは `products.coffee` と `products.sass` `products.coffee` し `products.sass` 。

に、**Rails**はコントローラ、ヘルパー、ビューのテストファイルをします。

Railsにそれらをスキップするようにするためにこれらのものをしたくないは、

`--no-`または `--skip`、このように

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

そしてジェネレータは `assets` と `helper` をスキップします

の `namespace` のコントローラをするがあるは、 `NAME` にし `namespace` 。

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

これで、コントローラは `app/controllers/admin/products_controller.rb` にされます

Railsは、なRESTfulコントローラをすることもできます。

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

```
rails generate scaffold_controller Product
```

RescueActiveRecord :: RecordNotFound with redirect_to

エラーページをするのではなく、リダイレクトを使ってRecordNotFoundをすることができます

```
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
```

オンラインでActionControllerをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2838/actioncontroller>

4: ActionMailer

き

アクションメーラーをすると、メーラーのクラスとビューをしてアプリケーションからメールを
できます。メーラーはコントローラーとによくしています。らはActionMailer :: Baseをし、app /
mailerにし、app / viewsにされるするビューをとっています。

メールのをにして、Webサーバーをらないようにすることをおめします。これは、`delayed_job`な
どのさまざまなサービスをじてうことができます。

Examples

メーラー

このでは、4つのなるファイルをします。

- ユーザーモデル
- ユーザーメーラ
- メールのhtmlテンプレート
- メールのプレーンテキストテンプレート

この、ユーザーモデルはメーラの`approved`メソッドをびし、`approved`した`post`をしますモデルの
`approved`メソッドは、コールバックによって、コントローラメソッドからびされるなど。に、メ
ーラーは、された`post` えば、タイトルからのをして、htmlまたはプレーンテキストのいずれかの
テンプレートからメールをします。デフォルトでは、メーラーは、メーラーのメソッドとじのテ
ンプレートをしますメーラー・メソッドとテンプレートのに「み」というがいているです。

user_mailer.rb

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

  def approved(post)
    @title = post.title
    @user = post.user
    mail(to: @user.email, subject: "Your Post was Approved!")
  end
end
```

user.rb

```
def approved(post)
  UserMailer.approved(post)
end
```

approved.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Post Approved</title>
  </head>
  <body>
    <h2>Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!</h2>
    <p>We look forward to your future posts!</p>
  </body>
</html>
```

approved.text.erb

```
Congrats <%= @user.name %>! Your post (#<%= @title %>) has been approved!
We look forward to your future posts!
```

しいメールをする

しいメールをするには、のコマンドをします

```
rails generate mailer PostMailer
```

これにより、 `app/mailers/post_mailer.rb` というの `app/mailers/post_mailer.rb` のテンプレートファイルがされます

```
class PostMailer < ApplicationMailer
end
```

HTMLとテキストの2つのレイアウトファイルもメールビューにされます。

ジェネレータをしたくないは、のメールをすることができます。 `ActionMailer::Base` からしていることをしてください

ファイルの

`ActionMailer` では、ファイルをすることもできます。

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

デフォルトでは、ファイルはBase64でエンコードされます。これをするには、`attachments`メソッドにハッシュをします。

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

インラインアタッチメントをすることもできます

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

ActionMailer コールバック

ActionMailerは3つのコールバックをサポートしています

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

Mailerクラスにこれらをする

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

`private`キーワードのこれらのメソッドをします

```
private
  def set_delivery_options
  end

  def prevent_delivery_to_guests
  end

  def set_business_headers
  end
end
```

スケジュールされたニュースレターをする

ニュースレターモデルをする

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

subl app/models/newsletter.rb

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

ニュースレターコントローラをする

```
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
```

その、 **create.html.erb**ビューを `new` にします。このファイルは、フッターにされるビューにされます。は **_form.html.erb** になります。

ファイルをから	に
<code>app / views / newsletters / create.html.erb</code>	<code>app / views / newsletters / _form.html.erb</code>

その、ルートをしします

```
subl app/config/routes.rb

resources :newsletters
```

で、メールをするためにするフォームをするがあります。

```
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>
```

```
<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 %>
```

その、フッターにします

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

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

、 - **letter_opener** - をインストールすると、メールをせずにデフォルトのブラウザでプレビューできます。つまり、でメールのをするはなく、ってののアドレスにテストメールをするはありません。

まず、gemをにし、bundleコマンドをしてインストールします。

```
subl your_project/Gemfile

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

に、でをします。

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

config.action_mailer.delivery_method = :letter_opener
```

さて、たちがくメーラーをするメーラーをしてください。ターミナル

```
rails generate mailer UserMailer newsletter_mailer
```

そして、 **UserMailer** ので、 **Newsletter Mailer** というメソッドをしなければなりません。これはのブログにをれるためにされ、レーキアクションでされます。これまでにしたブログのがあります。

```
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

end
```

その、メーラーテンプレートをします。

```
subl your_project/app/views/user_mailer/newsletter_mailer.html.erb

<p> Dear Followers: </p>
<p> Those are the latest 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>
```

メールをのプロセスとしてしたいので、メールをするためのRakeタスクをしましょう。
email_tasks.rakeというしいファイルをRailsアプリケーションのlib / tasksディレクトリにします

```
touch lib/taks/email_tasks.rake

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

send_digest_email ::とは、タスクをするにRailsをロードすることをし、タスクのUserMailerのよ
うなアプリケーションクラスにアクセスすることができます。

、rake -Tコマンドをすると、しくされたRakeタスクがリストされます。タスクをし、メールがさ
れているかどうかをして、すべてをテストします。

メーラメソッドがするかどうかをテストするには、rakeコマンドをします。

```
rake weekly_newsletter_email
```

では、**crontab**をしてスケジュールリングできるのがあります。そこで、cronジョブのとのためのなをするためにされる**Whenever Gem**をインストールします。

```
subl your_project/Gemfile

gem 'whenever', require: false
```

その、のコマンドをして、あなたのためのconfig / schedule.rbファイルをしませconfigフォルダがにプロジェクトにするり。

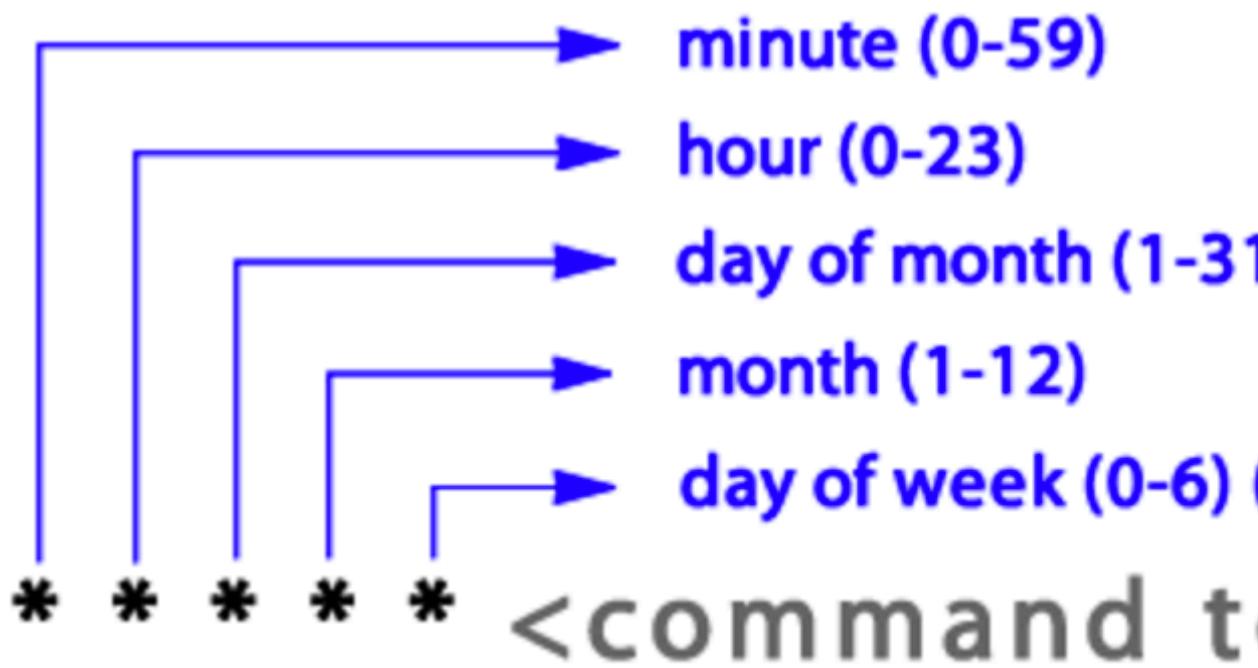
```
wheneverize .

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

スケジュールファイルので**CRON JOB**をし、**CRON JOB**をしてmailerメソッドをびすと、されたにいくつかのタスクをするがあります。この[リンク](#)でされているように、さまざまなのをできます。

```
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"
end
```

されたジョブをみむことができます。

```
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'
```

でテストをするには、**application.rb** プリンシパルファイルののをして、するモデルがどこにあるかをアプリケーションにらせてください。

```
subl your_project/config/application.rb

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

は**Capistrano V3**にしい**Cron Job**をサーバーにさせ、このタスクのをトリガーするトリガーをするために、のをするがあります。

```
subl your_project/Capfile

require 'whenever/capistrano'
```

CRON JOBがとアプリケーションのについてするをデプロイメント・ファイルにします。

```
subl your_project/config/deploy.rb

set :whenever_identifier, ->{ "#{fetch(:application)}_#{fetch(:rails_env)}" }
```

そして、ファイルのをしたら、**capistrano deploy** コマンドをします

```
cap production deploy
```

そしてあなたのがむもので、このファイルをするのでメーラーメソッドをするためにされ、カレンダーされました。

ActionMailer インターセプタ

アクションメーラーはインターセプターメソッドにフックをします。これらをするると、メールライフサイクルにびされるクラスをできます。

インターセプタクラスは**deliver_emailmessage**メソッドをするがあります。このメソッドは、メールがされるにびされ、メールがエージェントにたるにをえることをにします。あなたのクラスは、された**Mail :: Message**インスタンスになをえるべきです。

これは、がのユーザーではなくメールをするのにです。

actionmailer インターセプタの

```
# 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
```

オンラインでActionMailerをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2481/actionmailer>

5: ActiveRecord

ActiveModelは、ActiveRecordのモデルをのにするためにされました。これにより、ActiveRecordモデルだけでなく、どのオブジェクトでもActiveModelのことができます。

ActiveRecordオブジェクトには、デフォルトでこのがすべてまれています。

Examples

ActiveModel :: Validationsの

のルビーであっても、あらゆるオブジェクトをすることができます。

```
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
```

オンラインでActiveModelをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1773/activemodel>

6: ActiveRecord

Examples

モデルをでする

Scaffoldingをうのは、Railsをめてうやしいアプリケーションをるがいればですが、でされたコードをスリムするはありません。をりくなど。

モデルをするには、`app/models`にファイルをするだけのながあります。

もなモデルは、`ActiveRecord`、`ActiveRecord::Base`をしたクラスです。

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

モデルファイルは`app/models/`され、ファイルはクラスのにします。

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

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

クラスは、`ActiveRecord`のすべてのクエリメソッド、、コールバックなどをします。

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

するモデルのがすることをしてください。そうでないは、[を](#)してテーブルをできます

モデルをすることができ、それはのコマンドからのターミナルによるです

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

のコマンドでキーをモデルにりてることもできます

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

によるモデルの

Ruby on Railsは、`ActiveRecord`モデルののにできる`model`ジェネレータをします。 `rails generate model`をしてモデルをするだけです。

```
$ rails g model user
```

app/models モデルファイルにえて、ジェネレータはをします

- test/models/user_test.rb
- test/fixtures/users.yml
- db/migrate/XXX_create_users.rb のデータベースの

また、モデルをするとき、モデルのいくつかのフィールドをすることもできます。

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

これにより、なタイプのデータベースに email、sign_in_count、birthday というがされます。

をする

のテーブルのフィールドの/

のコマンドをしてをします。

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

これにより、 categories テーブルに title をするが categories ます。

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

に、マイグレーションをしてをすることもできます rails generate migration RemoveTitleFromCategories title:string

categories テーブルから title をするが categories ます。

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

にえば、のこのは :string はをするためにではありませんが、ロールバックにながされるのでです。

テーブルをする

のコマンドをしてをします。

```
rails g CreateUsers name bio
```

RailsはCreateからををするをし、りのほとしてされます。えられたは、ををする

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

コマンドでのがされておらず、のstringがされています。

ををする

のコマンドをしてをします。

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

Railsは、でJoinTableをつけることによってテーブルををするをします。のすべては、あなたがのに
えるフィールドのからされます。

```
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
```

なindexステートメントのコメントをし、りのステートメントをします。

サンプルのCreateJoinTableParticipationがテーブルのルールとしていることにしてください。この
テーブルにはCreateプレフィックスがありCreate。しかし、それはなcreate_tableしませんで
した。これは、ジェネレータ [ソースコード](#) がのリストののをするためです。

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

コールバックの

コールバックは、オブジェクトのライフサイクルののデータベースの、、、、、またはデータベ

ースからのロードのまたはにびされるメソッドです。

たとえば、から30にがれるリストがあるとします。

これをう1つのはのようなものです

```
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
```

コールバックのなメソッドはすべて、オブジェクトののびされるのとして、のとおりです。

オブジェクトの

- before_validation
- after_validation
- before_save
- around_save
- before_create
- around_create
- after_create
- after_save
- after_commit / after_rollback

オブジェクトの

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

オブジェクトをする

- before_destroy
- around_destroy
- after_destroy
- after_commit / after_rollback

`after_save`は、とのでされますが、マクロびしがされたにかかわらず、によりのコールバック `after_create`と `after_update`のにされます。

をしてをする

のためにに `has_and_belongs_to_many` は、でしてテーブルをすることができ `create_table` を。 `Tags`と `Projects` 2つのモデルがあり、 `has_and_belongs_to_many` をってそれらを `has_and_belongs_to_many` とします。のクラスのインスタンスをけるには、テーブルがです。

```
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
```

のは、このにうがあります。アルファベットにするモデルをにするがあります。 `Project`は `Tags`にするので、テーブルのは `projects_tags`です。

また、こののは、2つのモデルのインスタンスのけをルーティングすることであるため、こののレコードのIDはありません。これをするには、 `id: false` をし `id: false`

に、 `Rails`ののように、テーブルはモデルの々のモデルのでなければならぬが、テーブルのはでなければならぬ。

モデルのテスト

アクティブレコードモデルをコマンドラインインターフェイスでテストするのはです。の `app`ディレクトリにし、 `rails console` をして `Rails`コンソールをします。ここから、アクティブなレコードメソッドをデータベースでできます。

たとえば、 `name:string`カラムと `email:string`をつ `Users`テーブルをつデータベーススキーマがあるは、の命令をできます。

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

そのレコードをするには、の命令をします。

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

あるいは、これがあなたのレコードであれば、の命令をするだけのレコードをできます

```
User.first
```

モデルインスタンスをしてをする

User モデルがあるとします

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

id = 1 ユーザーの first_name と last_name をするには、のコードをします。

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

update をびすと、のトランザクションのされたをしようとし false。し false は true し、したは false し true。

オンラインで ActiveRecord をむ <https://riptutorial.com/ja/ruby-on-rails/topic/828/activerecord>

7: ActiveRecord トランザクション

トランザクションはブロックで、SQLはすべてが1つのアトミックアクションとしてすることができればです。なほ、きしがしたにのみをつことができる2つののであり、そのもあります。トランザクションは、データベースのをし、プログラムエラーやデータベースのからデータをします。したがって、には、にされるか、まったくされなければならないステートメントがあるは、トランザクション・ブロックをするがあります。

Examples

な

えば

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

このでは、ダビデのおしかっておらず、ももをしなければ、それをメアリーにえます。は、トランザクションがされるのにデータベースをすROLLBACKをします。ただし、オブジェクトのインスタンスデータはトランザクションのにされません。

1つのトランザクションのなるActiveRecordクラス

トランザクションクラスメソッドはのActiveRecordクラスでびされますが、トランザクションブロックのオブジェクトはすべてそのクラスのインスタンスであるはありません。これは、トランザクションがモデルではなくデータベースのであるためです。

このでは、がAccountクラスでびされても、レコードはトランザクションでされます。

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

トランザクションメソッドは、モデルインスタンスメソッドとしてもできます。たとえば、のよようにすることもできます。

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

のデータベース

トランザクションはデータベースでします。クラスのデータベースがある、そのトランザクションはそれらののをしません。1つのは、モデルをしたクラスでトランザクションをすることです。

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

これはなソリューションですが、にしたトランザクションはActiveRecordのです。

とはにトランザクションにラップされます

#saveと#destroyのがトランザクションにラップされています。これにより、やコールバックでうことは、のカバーののにされます。したがって、をしてトランザクションがするをチェックするか、after_*コールバックをむロールバックのコールバックでをさせることができます。

として、データベースへののは、がするまでにはえません。たとえば、after_saveエンジンのインデックスをしようとする、インデクサーはされたレコードをしません。after_commitコールバックは、がコミットされるとトリガーされるのコールバックです。

コールバック

トランザクションのコミットとロールバックにするコールバックには、after_commitとafter_rollback 2があります。

after_commitコールバックは、トランザクションがコミットされたにトランザクションでまたはされたすべてのレコードでびされます。after_rollbackコールバックは、トランザクションまたはセーブポイントがロールバックされたに、トランザクションでまたはされたすべてのレコードでびされます。

これらのコールバックは、データベースがなにあるときにのみコールバックがされることがされるため、のシステムとのやりとりにちます。たとえば、after_commitは、トランザクションからキャッシュをクリアすると、データベースがされるにキャッシュがされるがあるため、キャッシュをクリアするフックをれるのにしています。

トランザクションのロールバック

ActiveRecord::Base.transactionは、ActiveRecord::Rollbackをしてなロールバックをのなとします。、をさせると、.transactionメソッドはデータベーストランザクションをロールバックし、をします。しかし、ActiveRecord::Rollbackをさせると、データベーストランザクションはをすことなくロールバックされます。

たとえば、コントローラでトランザクションをロールバックするには、のようになります。

```
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
```

オンラインでActiveRecordトランザクションをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4688/activerecordトランザクション>

9: ActiveRecordのロック

Examples

なロック

```
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
```

なロック

```
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
```

オンラインでActiveRecordのロックをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3866/activerecordのロック>

10: ActiveRecordの

パラメーター

の	
:primary_key	キー
:string	いデータ。のlimitオプションをします。
:text	よりいテキスト。バイトのlimitオプションをします。
:integer	。バイトのlimitオプションをします。
:bigint	よりきい
:float	く
:decimal	の10。 precisionとscaleオプションがです。
:numeric	precisionとscaleオプションがです。
:datetime	/のDateTimeオブジェクト。
:time	オブジェクト。
:date	のDateオブジェクト。
:binary	バイナリデータ。バイトのlimitオプションをします。
:boolean	ブール

- ほとんどのファイルはdb/migrate/ディレクトリにあります。それらは、ファイルのにあるUTCタイムスタンプ YYYYMMDDHHMMSS_create_products.rbによってされます。
- rails generateコマンドは、 rails gくすることができます。
- a :typeがフィールドにされない、デフォルトはになります。

Examples

のをする

のをまたはにするには、 db:migrate:upまたはdb:migrate:downます。

のをする

5.0

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

5.0

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

のをする

5.0

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

5.0

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

のコマンドのバージョンは、のファイルにあるのです。たとえば、
20160515085959_add_name_to_users.rbにするには、バージョンとして20160515085959をします。

をする

studentsとcoursesにテーブルをするには、のコマンドをします。

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

これにより、のがされます。

```
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
```

なるでの

testでするには、のシェルコマンドをします。

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

5.0

Rails 5.0から、rakeではなくrailsをすることができます

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

テーブルにしいをする

`users` に `name` をするには、の コマンド をします。

```
rails generate migration AddNameToUsers name
```

これにより、のがされます。

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

マイグレーションのが `AddXXXXToTABLE_NAME` で、データ・タイプののリストがく、されたマイグレーションには `add_column` ステートメントがまれます。

インデックスきのしいをする

しい `email` を `users` にするには、の コマンド をします。

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

これにより、のがされます。

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

テーブルからのをする

`users` テーブルからの `name` をするには、の コマンド をします。

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

これにより、のがされます。

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

マイグレーションのが`RemoveXXXFromYYY`、データ・タイプののリストがき、されたマイグレーションにはな`remove_column`ステートメントがまれます。

データ:`string` を`remove_column`パラメータとしてするはありませんが、これをおめします。データがされていない、はにせません。

にをする

`team`へのを`users`テーブルにするには、のコマンドをします。

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

これにより、のがされます。

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

そのによって、`users`に`team_id`がされます。

されたにな`index`と`foreign_key`をするは、コマンドを`rails generate migration AddTeamRefToUsers team:references:index`。これにより、のがされます。

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

あなたがRailsのものごにをけたいは、あなたののにのをしますあなたがびしたいかもしれない`User`
`Post`として`Author`で`Post`テーブル

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

しいテーブルをする

カラム`name`と`salary`をしてしい`users`テーブルをするには、のコマンドをします。

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

これにより、のがされます。

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

マイグレーションが`CreateXXX`のデータとそれにクデータ・タイプののリストである、リストされたをつ`XXX`をするマイグレーションがされます。

テーブルにのをする

テーブルにのカラムをするには、`rails generate migration`コマンドを`rails generate migration`ときに、`field:type`ペアをスペースで`field:type`。

なはのとおりです。

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

たとえば、のようにすると、`name`、`salary`、`email`フィールドが`users`テーブルにされます。

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

のがされます。

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

の

コマンドをする

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

ターゲットバージョンをすると、されたバージョンにするまで、なマイグレーション`up`、`down`、`change`がされます。ここで、`version number`はのファイルのです。

5.0

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

5.0

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

のロールバック

のを `rollback` するには、 `change` メソッドを `up` にすか、 `down` メソッドを `down` ます。コマンドをする

5.0

```
rake db:rollback
```

5.0

```
rails db:rollback
```

3のをロールバックする

5.0

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

5.0

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

`STEP` は、 `up` の `STEP` をします。

すべてのをロールバックする

5.0

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

5.0

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

テーブルの

ったスキーマをつをした、とそのプロパティーをするものは `change_table` です。のをしてください。

```
change_table :orders do |t|  
  t.remove :ordered_at # removes column ordered_at  
end
```

```
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
```

のは、テーブルの`orders`します。はのとおりです。

1. `t.remove :ordered_at`は、テーブル`orders`から`ordered_at`カラムをします。
2. `t.string :skew_number`は、`orders`に`skew_number`というのしいをします。
3. `t.index :skew_number`は`orders`テーブルの`skew_number`カラムにインデックスをします。
4. `t.rename :location, :state`リネーム`location`におけるカラム`orders`にテーブルを`state`。

テーブルにのをする

しいのの`email`を`users`にするには、のコマンドをします。

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

これにより、のがされます。

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

のをする

でRailsののをするには、のコマンドをします。

```
rails g migration change_column_in_table
```

これにより、`db/migration`ディレクトリにしいファイルがされますまだしていない。このファイルには、タイムスタンプと、のをむファイルのがけられます。

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

[Railsガイド](#) - の

よりくよりな

のコードは、ユーザーがをロールバックすることをします。このをするには、`change`メソッドを々の`up`メソッドと`down`メソッドに`down`ます。

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

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

のやりし

`redo` コマンドをして、ロールバックしてからすることができます。これは、に、 `rollback` タスクと `migrate` タスクをみわせたショートカットです。

コマンドをする

5.0

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

5.0

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

`STEP` パラメーターをすると、のバージョンにすることができます。

たとえば、3つのするには

5.0

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

5.0

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

デフォルトのを

のでは、 `admin` を `users` にし、そのにデフォルト `false` します。

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

PostgreSQLなどのきなテーブルでは、デフォルトのでかかるがあります。これは、しくされたのデフォルトでをするがあるためです。これをしのダウンタイムをらすために、を3つのステップにけることができます。

1. との `add_column -migration` をしますが、デフォルトはしません

- アプリケーションの rake タスクまたはコンソールにをデプロイしてします。アプリケーションが、そのにしい/されたのデータをすでにきんでいることをしてください。
- の `change_column` マイグレーションをします。これにより、そののデフォルトがのデフォルトにされます

ヌルをする

テーブルの `nullable` をするには、のように `:nullable` パラメータをにします。

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

ステータスの

することでのをできます

3.05.0

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

5.0

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

はのようになります。

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 *****

`status` フィールドの `up` は、が `down` ことをし、 `down` は、をするがあることをします。

hstore をする

`hstore` カラムは、をするのにです。をにしたは PostgreSQL データベースでできます。

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

```
t.timestamps
end
end
end
```

をする

は、ツリーをするのにです。これは、に `add_reference` してできます。

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

キーは `pages_id` ます。キーのをするは、にをしてからをするがあります。

```
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
```

をする

`array` は PostgreSQL でサポートされています。 Rails はに PostgreSQL を Ruby にします。もまたです。

`array` をむをします。

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

のテーブルに `array` カラムをする

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

`array` のインデックスをする

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

のデータに **NOT NULL** をする

キー `company_id` を `users` テーブルにし、そのテーブルに `NOT NULL` をしたいとします。 `users` にデータがあるは、のでこれをうがあります。

```
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

    # 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
```

オンラインでActiveRecordのをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1087/activerecord/>

11: ActiveRecord

Examples

のの

これは、のみのをします。

```
class Player < ApplicationRecord
  validates :points, numericality: true
  validates :games_played, numericality: { only_integer: true }
end
```

これに `:only_integer`、このヘルパーは、けれなにをすためののオプションもくれます

- `:greater_than` - がされたよりきくなければならないことをします。このオプションのデフォルトのエラーメッセージは "{count}よりきいがあります"です。
- `:greater_than_or_equal_to` - がされたでなければならぬことをします。このオプションのデフォルトのエラーメッセージは、 "{count}でなければなりません"です。
- `:equal_to` - がされたとしくなければならぬことをします。このオプションのデフォルトのエラーメッセージは "{count}としくなければなりません"です。
- `:less_than` - がされたよりさくなければならぬことをします。このオプションのデフォルトのエラーメッセージは "{count}よりさいがあります"です。
- `:less_than_or_equal_to` - がされたでなければならぬことをします。このオプションのデフォルトのエラーメッセージは、 "{count}でなければなりません"です。
- `:other_than` - がされたでなければならぬことをします。このオプションのデフォルトのエラーメッセージは "{count}であるがあります"です。
- `:odd` - `true`にされている、はでなければならぬことをします。このオプションのデフォルトのエラーメッセージは、 "でなければなりません"です。
- `:even` - `true`にされている、はでなければならぬことをします。このオプションのデフォルトのエラーメッセージは「ずするがあります」です。

デフォルトでは、では`nil`はできません。 `allow_nil`オプションをしてすることができま

のをす

このヘルパーは、オブジェクトのののがであることをします。

```
class Account < ApplicationRecord
  validates :email, uniqueness: true
end
```

をすためにされる1つのをすためにできる `a :scope` オプションがあります。

```
class Holiday < ApplicationRecord
  validates :name, uniqueness: { scope: :year,
    message: "should happen once per year" }
end
```

がとをやるかどうかをやるためにできる`:case_sensitive`オプションもあります。このオプションのデフォルトは`true`です。

```
class Person < ApplicationRecord
  validates :name, uniqueness: { case_sensitive: false }
end
```

のの

このヘルパーは、されたがでないことをします。

```
class Person < ApplicationRecord
  validates :name, presence: true
end

Person.create(name: "John").valid? # => true
Person.create(name: nil).valid? # => false
```

`absence`ヘルパーをして、されたがしないことをできます。それは`present?`して`present?`メソッドをして、ゼロまたはのをチェックします。

```
class Person < ApplicationRecord
  validates :name, :login, :email, absence: true
end
```

が`boolean`、のはできませんは`false`にしてではありません。これをめるには、インクルードをします。

```
validates :attribute, inclusion: [true, false]
```

のスキップ

をスキップするは、のメソッドをします。これらのメソッドは、オブジェクトがであってもオブジェクトをデータベースにします。

- デクリメント
- `decrement_counter`
- インクリメント
- `increment_counter`
- トグル
- タッチ
- `update_all`
- `update_attribute`

- `update_column`
- `update_columns`
- `update_counters`

`validate` をとしてしてするにをスキップすることもでき `save`

```
User.save(validate: false)
```

のさの

```
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
```

なさオプションはのとおりです。

- `:minimum` - のさはされたさをすることはできません。
- `:maximum` - はされたさをえることはできません。
- `:in` または `:within` - のさは、されたにめるがあります。このオプションのはでなければなりません。
- `:is` - のさは、されたとしくなければなりません。

グループの

ので1つのをすとなことがあります。これは `with_options` をしてにできます。

```
class User < ApplicationRecord
  with_options if: :is_admin? do |admin|
    admin.validates :password, length: { minimum: 10 }
    admin.validates :email, presence: true
  end
end
```

`with_options` ブロックのすべてのバリデーションは、のにをにします `is_admin`

カスタム

`ActiveModel::Validator` または `ActiveModel::EachValidator` するしいクラスをして、のをできます。どちらのもていますが、なるでします。

`ActiveModel::Validator` **および** `validates_with`

レコードをとしてけり、そのレコードにして `validate` する `validate` メソッドをします。その、モデルのクラスで `validates_with` をします。

```
# app/validators/starts_with_a_validator.rb
class StartsWithAValidator < ActiveRecord::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 および validate

のパラメータでの `validate` メソッドをしてしいバリデーターをするは、 `ActiveModel::EachValidator` からするクラスをし、 `record`、 `attribute`、 および `value` 3つのを `validate_each` メソッドをし `value`。

```
class EmailValidator < ActiveRecord::EachValidator
  def validate_each(record, attribute, value)
    unless value =~ /\A(?:[^\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\z/i
      record.errors[attribute] << (options[:message] || 'is not an email')
    end
  end
end

class Person < ApplicationRecord
  validates :email, presence: true, email: true
end
```

Railsガイドの。

のフォーマットをします。

のが `format` と `with` オプションをすることをするを `with` ます。

```
class User < ApplicationRecord
  validates :name, format: { with: /\A\w{6,10}\z/ }
end
```

をし、そのをにして `with`: オプションにすこともできます。になのがかもしれません

```
PHONE_REGEX = /\A(?:\d{3})\d{3}-\d{4}\z/
validates :phone, format: { with: PHONE_REGEX }
```

デフォルトのエラーメッセージ `is invalid` です。これは `:message` オプションでできます。

```
validates :bio, format: { with: /\A\D+\z/, message: "Numbers are not allowed" }
```

もし、 `が` としないようにすることもできます `without`: オプション

のをする

`inclusion:` ヘルパーをしてがにまれているかどうかをできます。 `:in` オプションとそのエイリアス `:in :within` なのセット。

```
class Country < ApplicationRecord
  validates :continent, inclusion: { in: %w(Africa Antartica Asia Australia
                                           Europe North America South America) }
end
```

がにまれていないかどうかをべるには、 `exclusion:` ヘルパーをします

```
class User < ApplicationRecord
  validates :name, exclusion: { in: %w(admin administrator owner) }
end
```

き

によっては、のでのみレコードをするがあるがあります。

```
class User < ApplicationRecord
  validates :name, presence: true, if: :admin?

  def admin?
    conditional here that returns boolean value
  end
end
```

あなたがきでにさいは、`Proc`をうことができます

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: Proc.new { |user| user.last_name.blank? }
end
```

ののは、のを `unless` できます。

```
class User < ApplicationRecord
  validates :first_name, presence: true, unless: Proc.new { |user| user.last_name.present? }
end
```

`instance_eval` でされるをすこともでき `instance_eval`

```
class User < ApplicationRecord
  validates :first_name, presence: true, if: 'last_name.blank?'
end
```

の

2つのテキストフィールドがしをにけるがあるは、これをしてください。たとえば、メールアドレス

またはパスワードをすることができます。このによって、`_confirmation` けてする `_confirmation` があるフィールドのをとするがされます。

```
class Person < ApplicationRecord
  validates :email, confirmation: true
end
```

メモこのチェックは、`email_confirmation` が `nil` でないにのみされます。

をするには、のチェックをずしてください。

```
class Person < ApplicationRecord
  validates :email,          confirmation: true
  validates :email_confirmation, presence: true
end
```

ソース

on オプションをう

`:on` オプションをすると、がいつわれるかをできます。すべてのビルトインヘルパーのデフォルトのは、しいレコードをするときとするときなので、にされます。

```
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
```

オンラインで ActiveRecord をむ <https://riptutorial.com/ja/ruby-on-rails/topic/2105/activerecord>

12: ActiveRecord インターフェース

き

ActiveRecordは、ビジネスデータとロジックをすシステムのレイヤーであるMVCのMです。リレーショナルデータベースシステムのテーブルへのアプリケーションのなオブジェクトをするがObject Relational MapperのORMです。

ActiveRecordは、データベースのクエリをし、ほとんどのデータベースシステムとがあります。どのデータベースシステムをしていても、ActiveRecordメソッドはにじです。

Examples

.where

whereメソッドはのActiveRecordモデルででき、されたにするレコードのセットをデータベースにいわせることができます。

whereメソッドは、キーがモデルがすテーブルのにするハッシュをくれます。

なとして、のモデルをします。

```
class Person < ActiveRecord::Base
  #attribute :first_name, :string
  #attribute :last_name, :string
end
```

ののSvenすべてのをつけるには

```
people = Person.where(first_name: 'Sven')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven'"
```

SvenとSchrodingerをつすべてのをつけるには

```
people = Person.where(first_name: 'Sven', last_name: 'Schrodinger')
people.to_sql # "SELECT * FROM people WHERE first_name='Sven' AND last_name='Schrodinger'"
```

のでは、sqlは、first_nameとlast_nameがしたにのみレコードがされることをしています。

ORきクエリ

first_name == 'Bruce' OR last_name == 'Wayne'レコードをするには

```
User.where('first_name = ? or last_name = ?', 'Bruce', 'Wayne')
# SELECT "users".* FROM "users" WHERE (first_name = 'Bruce' or last_name = 'Wayne')
```

。あり

のActiveRecordモデルの`where`メソッドをして、`WHERE column_name IN (a, b, c, ...)`というのSQLをできます。これは、としてをすことによつてされます。

なとして、のモデルをします。

```
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')"
```

に`nil`がまれている、そのが`null`かどうかをするためにSQLがされ`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"
```

スコープ

スコープは、ActiveRecordモデルでみのフィルタとしてします。

スコープは`scope`クラスメソッドをしてされます。

なとして、のモデルをします。

```
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
```

スコープはモデルクラスからびすことができます

```
minors = Person.minors
```

スコープはできます。

```
peters_children = Person.minors.with_last_name('Peters')
```

where

メソッドやのクエリーメソッドもすることができます

```
mary_smith = Person.with_last_name('Smith').where(first_name: 'Mary')
```

では、スコープはなクラスのメソッドのなです。たとえば、これらのメソッドはにじです。

```
scope :with_last_name, ->(name) { where(name: name) }

# This ^ is the same as this:

def self.with_last_name(name)
  where(name: name)
end
```

デフォルトスコープ

モデルのすべてののデフォルトスコープをします。

`scope`メソッドとクラスメソッドのにはないがあり`scope`。 `scope`スコープは、そのロジックが`nil`をすでもに `ActiveRecord::Relation` します。しかし、クラスメソッドはそのようなセーフティネットをたず、かをすとをるがあります。

どこにもない

`where`はしてすることができ`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')
```

ActiveRecord 4.0でサポートされています。

`.order`をして**ActiveRecord**クエリをすることができます

```
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">]
```

されていないは、にべえられます。のようにしてすることができます

```
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`はもくれます

```
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">]
```

はこのSQLであるため、だけでなくテーブルもできます。の`role`によって`users`をするは、の`users`にします。

```
Class User < ActiveRecord::Base
  belongs_to :role
end

Class Role < ActiveRecord::Base
  has_many :users
end

User.includes(:role).order("roles.name ASC")
```

`order`スコープは、`Arel`ノードをけることもできます。

```
User.includes(:role).order(User.arel_table[:name].asc)
```

ActiveRecord Bang メソッド

したに`false`ではなくをさせるために**ActiveRecord**メソッドがなは、することができ!らへ。これはとてもです。あなたがしない、いくつかの/はまえにくいのでそれらのに。サイクルでこれをして、**ActiveRecord**コードをこのようにくことで、とをくことをおめします。

```
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
```

bang ! をける**ActiveRecord**メソッドはのとおりです。

- `.create!`
- `.take!`
- `.first!`
- `.last!`
- `.find_by!`
- `.find_or_create_by!`
- `#save!`
- `#update!`
- すべてのARダイナミックファインダ

`.find_by`

`find_by`をして、テーブルののフィールドでレコードをつけることができます。

したがって、`first_name` をつ `User` モデルをすると、のことができます。

```
User.find_by(first_name: "John")
#=> #<User id: 2005, first_name: "John", last_name: "Smith">
```

`find_by` がデフォルトでをスローしないことを `find_by` で `find_by` てください。がであれば、 `find` わりに `nil` をし `find` 。

がなは、 `find_by!` することができます `find_by!` それは、 `find` ような `ActiveRecord::RecordNotFound` エラーをさせ `find` 。

。すべて

くのレコードをすばやくするがある、 **ActiveRecord** は `.delete_all` メソッドをします。モデルで `び`す、そのテーブルのすべてのレコードを `する`、またはコレクションを `する`。ただし、`.delete_all` はオブジェクトをインスタンスしないため、コールバックはされません `before_*` および `after_destroy` は `after_destroy` ません。

```
User.delete_all
#=> 39 <-- .delete_all return the number of rows deleted

User.where(name: "John").delete_all
```

ActiveRecord のとをしない

ActiveRecord モデルでの `を`するがあるは、 `LIKE` または `ILIKE` をするようにされるかもしれませんが、データベースエンジンではできません。に、に `ダウンケーシング` または `アップケーシング` になることは、パフォーマンスの `を`きこすがあります。

ActiveRecord のとなる `Arel` `matches` メソッドを `する`と、なでこれを `う`ことができます

```
addresses = Address.arel_table
Address.where(addresses[:address].matches("%street%"))
```

`Arel` は、されたデータベースエンジンにな `LIKE` または `ILIKE` を `します`。

とのレコードを `する`

`Rails` は、データベースから `first` と `last` レコードを `する` にな `を`っています。

`users` テーブルから `first` レコードを `する` には、の `コマンド` を `する` があります。

```
User.first
```

それはの `sql` クエリを `します`

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` ASC LIMIT 1
```

そして、のをすでしょう

```
#<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 >
```

`users` テーブルから `last` レコードをするには、の コマンドをするがあります。

```
User.last
```

それはの `sql` クエリをします

```
SELECT `users`.* FROM `users` ORDER BY `users`.`id` DESC LIMIT 1
```

そして、のをすでしょう

```
#<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 >
```

とのメソッドにをすと、**LIMIT** クエリがされ、オブジェクトのがされます。

```
User.first(5)
```

これは、の `sql` クエリをします。

```
SELECT "users".* FROM "users" ORDER BY "users"."id" ASC LIMIT 5
```

そして

```
User.last(5)
```

これは、の `sql` クエリをします。

```
SELECT "users".* FROM "users" ORDER BY "users"."id" DESC LIMIT 5
```

`.group` と `.count`

たちは `Product` モデルを持っており、それらの `category` グループしたいとえてい `category`。

```
Product.select(:category).group(:category)
```

これにより、のようにデータベースがされます。

```
SELECT "product"."category" FROM "product" GROUP BY "product"."category"
```

グループされたフィールドもされていることをします。グループは、`categories`このをえるのに
にです。

```
Product.select(:category).group(:category).count
```

クエリがすように、すべてのレコードをにし、コードでカウントするよりもはるかにな、カウン
トにデータベースをします

```
SELECT COUNT("products"."category") AS count_categories, "products"."category" AS  
products_category FROM "products" GROUP BY "products"."category"
```

.distinctまたは**.uniq**

からをするは、`.distinct()`できます。

```
Customers.select(:country).distinct
```

これは、のようにデータベースにします。

```
SELECT DISTINCT "customers"."country" FROM "customers"
```

`.uniq()`はじをちます。 Rails 5.0ではされ、バージョン5.1のRailsからされます。そのは、`unique`
というはなるものとじをたず、をくがあるからです。さらに、`distinct`はSQLにい。

`joins()`すると、テーブルをのモデルにできます。えは、

```
User.joins(:posts)
```

のSQLクエリをします。

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id"
```

テーブルにすると、そのテーブルにアクセスできます。

```
User.joins(:posts).where(posts: { title: "Hello world" })
```

にしてください。あなたのが`:has_many`、`joins()`はにするがあります。それのは、をします。

ネストされた`joins`

```
User.joins(posts: :images).where(images: { caption: 'First post' })
```

それはをする

```
"SELECT "users".* FROM "users" INNER JOIN "posts" ON "posts"."user_id" = "users"."id" INNER  
JOIN "images" ON "images"."post_id" = "images"."id"
```

まれるもの

`ActiveRecord` with `includes` は、されたすべてのアソシエーションがのクエリをしてロードさ `includes` ようにします。したがって、けられたテーブルをつデータのテーブルをクエリすると、のテーブルがメモリにロードされます。

```
@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 })` は、をみまらずにきのだけをメモリにみみます。ネストされたけにするがないは、`joins` し `joins` 。

```
@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
```

リミットとオフセット

`limit` をしてフェッチするレコードのをし、`offset` をしてレコードのをしてスキップしてレコードをすことができます。

えば

```
User.limit(3) #returns first three records
```

これは、のSQLクエリをします。

```
"SELECT `users`.* FROM `users` LIMIT 3"
```

のクエリではオフセットがされていないので、の3つのレコードがされます。

```
User.limit(5).offset(30) #returns 5 records starting from 31th i.e from 31 to 35
```

これは、のSQLクエリをします。

```
"SELECT `users`.* FROM `users` LIMIT 5 OFFSET 30"
```

オンラインでActiveRecordインターフェースをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2154/activerecordインターフェース>

13: ActiveSupport

ActiveSupportは、のRailsフレームワークでされているツールのユーティリティです。

これらのツールをするなの1つは、RubyのネイティブタイプをMonkeypatchingすることです。これらをコアとびます。

Examples

コアアクセス

Stringat

オブジェクトのをします。 `String#[]` ジインタフェースです。

```
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)  # => ""
```

Stringfrom

されたからのまでののをします。

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

String to

のからされたまでののをします。

がのは、のからえられます。

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

`from`と`to`はにできます。

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

の

の、またはされたをのさまでします。

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

ストリング

の、またはにえるのからされたをします。

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

コアからへの

ストリングto_time

をTimeにします。 `form` パラメータは、`:utc` または `:local` いずれか `:utc`。 デフォルトは `:local` です。

```
"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
```

ストリングto_date

をDateにします。

```
"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
```

to_datetime

をDateTimeにします。

```
"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
```

コアの

String#include?のString#include?

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

コアフィルタ

squish

またはののないされたのバージョンをし、のしたすべてののをのにします。なバージョンsquish!インスタンスにしてにします。

ASCIIとUnicodeののをいます。

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

remove

すべてのパターンがされたしいをします。なバージョンremove!されたにしてします。

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

ストリングりめ

がさよりもい、されたさでりてられたのコピーをします。

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

またはをす:separatorなりでりてるための:separator

```
'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..."
```

truncate_words

えられたのにりてられたをします。

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

またはをして、なるのりをする

```
'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..."
```

のは:omissionデフォルトは "... "にきえられます。

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

ストリングstrip_heredoc

heredocsのインデントをします。もインデントされていないでないがされ、のがされます。

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

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

ユーザは、

```
This command does such and such.
```

```
Supported options are:
```

```
-h          This message
```

```
...
```

コア String Inflection

のをします。オプションで `count` パラメータをとり、`count count == 1` はをします。ののための `locale` パラメータもつけられ `locale`。

```
'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"
```

singularize

のをします。オプションの `locale` パラメータを付けます。

```
'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"
```

constantize

でされたでされたをつけようとします。が `NameError` にない、またはされていない、`NameError` します。

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

String#safe_constantize

`constantize` しますが、`NameError` を `NameError` せるわりに `nil` をします。

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

String#camelize

は、デフォルトで `UpperCamelCase` にをし、`:lower` param はわりに `lowerCamelCase` にとしてえられています。

エイリアス `camelcase`

/ を `::` にし、パスをにするのにです。

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

String#titleize

すべてのをにし、ののをしてよりえのいタイトルをします。

エイリアス `titlecase`

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

String#underscore

のからのアンダースコアをします。 `camelize` の。

`underscore` は `::` `to` / をしてをパスにします。

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

String#dasherize

のをダッシュでできます。

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

demodulize

のからモジュールのをします。

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

deconstantize

のからものセグメントをします。

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

Stringparameterize

のをきえて、「pretty」URLのとしてできるようにします。

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

:preserve_case をして、ののとを:preserve_case ます。

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

parameterize のには、ActiveRecordモデルのto_paramメソッドをオーバーライドして、よりto_param URLスラッグをサポートすることです。

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

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

tableize

モデルからテーブルへのRailsのようなテーブルのをします。ののをPluralizeします。

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

Stringclassify

テーブルからモデルへのRailsのようにのテーブルからクラスをします。

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

humanize

のをにし、アンダースコアをスペースにし、するは_idます。

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

ストリングupcase_first

のだけをにします。

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

foreign_key

クラスからキーをします。とidに_をできないようにするには、false paramをしfalse。

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

オンラインでActiveSupportをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4490/activesupport>

14: CanCanによる

き

CanCanは、ユーザーのからりされたRailsにするなです。すべてののは1つのにされます。

CanCanをするに、をするかユーザーをすることをれないでください。CanCanののをするには、ユーザーをします。

Examples

CanCanをいめる

CanCanは、のリソースへのユーザーアクセスをするRuby on Railsのなライブラリです。のCanCanCanは、んだプロジェクトCanCanのきです。

パーミッションはAbilityクラスでされ、コントローラ、ビュー、ヘルパー、またはコードののからできます。

アプリケーションにサポートをするには、CanCanCan gemをGemfileにしGemfile

```
gem 'cancancan'
```

に、クラスをします。

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

  def initialize(user)
    end
  end
end
```

に、load_and_authorize_resourceをしてをチェックして、されたモデルをコントローラにロードします。

```
class ArticlesController < ApplicationController
  load_and_authorize_resource

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

authorize! をするか、をさせる

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

can? オブジェクトがコントローラ、ビュー、またはヘルパーのアクションにしてされているかどうかをチェックする

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

これは、されたユーザーが `current_user` メソッドによってされていることをとしています。

の

は、`can` と `cannot` メソッドをして `Ability` クラスでされます。なのためのコメントきのをえてみましょう。

```
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
```

のをう

のがえめると、ファイルのをうのがますますになります。

これらのをするためのものは、こののように、をのあるにすることです。

```
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

```

このクラスがにきくなると、のクラスにして、のようなさまざまなをできます。

```

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

  def initialize(user)
    self.merge Abilities::Everyone.new(user)

    if user
      if user.admin?
        self.merge Abilities::Admin.new(user)
      else
        self.merge Abilities::Authenticated.new(user)
      end
    else
      self.merge Abilities::Guest.new(user)
    end
  end
end

```

これらのクラスをのようになります。

```

# app/models/abilities/guest.rb

```

```
module Abilities
  class Guest
    include CanCan::Ability

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

Abilities::Authenticated、 Abilities::Adminまたはのものをします。

すばやくをテストする

クラスがなをえているかどうかをくテストしたいは、コンソールやのコンテキストでrailsがロードされたでをすることができます。

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

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

オンラインでCanCanによるをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3021/cancan>による

15: DeviseをしてApiをする

き

DeviseはRailsのソリューションです。さらにめるに、APIにするなメモをしたいといいます。したがって、APIはセッションをしませんステートレスです。これは、にをし、それのをとしないことをします。つまり、サーバーにするたびにシステムがするために、すべてのAPIでのをし、Deviseにのをしないようにするがあります。

Examples

そこで、まず、ルールプロジェクトとセットアップデバイスを行います

ルールアプリケーションをする

```
rails new devise_example
```

すぐリストにする

railsプロジェクトのルートにある 'Gemfile' というファイルをつけることができます

にbundle installします

に、ジェネレータをするがあります。

```
rails generate devise:install
```

コンソールでは、それにくはほとんどありません。

モデルをする

```
rails generate devise MODEL
```

に、 rake db:migrate します rake db:migrate

はDevise Gemをごください。

トークン

トークンは、のトークンをつユーザーをするためにされます。まず、ロジックをめるに、auth_token フィールドをDeviseモデルにするがあります

したがって、

```
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
```

に、 rake db:migrate します rake db:migrate

auth_token をってをうように auth_token

app/controllers/application_controllers.rb

にこのに

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

これは、レールアプリケーションがhtmlとjsonのであるのにちます

その、

```
protect_from_forgery with: :null
```

これをします :null たちはセッションをっていないので、 :null です。

ここで、 application_controller にメソッドをします

したがって、デフォルトでは、 Devise はメールをのフィールドとしてし、カスタムフィールドもできます。この、 user_email と auth_token をしてされます。

```
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
```

のコードはにあなたのロジックに基づいています。はのをしようとしています。

のコードの6では、 store: false をしていることがわかります store: false これはリクエストでセッションをできないようにするため、

オンラインで Devise をして Api をするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9787/devise>
をして api をする

16: GoogleMaps と Rails の

Examples

レイアウトヘッダーに **Google** マップの **javascript** タグをする

Google マップを [ターボリンク](#) でしくさせるには、javascript タグをビューにめるのではなく、レイアウトヘッダーにします。

```
# app/views/layouts/my_layout.html.haml
!!!
%html{:lang => 'en'}
  %head
    - # ...
    = google_maps_api_script_tag
```

`google_maps_api_script_tag` は、ヘルパーでもよくされています。

```
# 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
```

あなたは、あなたのアプリケーションを Google にし、[Google の API コンソール](#) にあなたの API キーをすることができます。Google には、[Google マップの javascript API の API キーをリクエストするについてのガイド](#)があります。

API キーは `secrets.yml` ファイルにされて `secrets.yml` ます

```
# config/secrets.yml
development:
  google_maps_api_key: '...'
  # ...
production:
  google_maps_api_key: '...'
  # ...
```

`.gitignore` ファイルに `config/secrets.yml` をすることを **な**しないでください。リポジトリに API キーをコミットしないようにしてください。

モデルをジオコードする

ユーザーおよびまたはグループにプロフィールがあり、アドレスマップのフィールドをGoogleマップにするとします。

```
# 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
```

すなわち、をジオコードするためのらしい `longitude` と `latitude` ある [ジオコード](#)。

Gemfile し、 `bundle` をしてインストールします。

```
# Gemfile
gem 'geocoder', '~> 1.3'
```

`latitude` と `longitude` データベースをして、データベースのをします。これは、をとするたびにジオコーディングサービスをするよりです。それはく、クエリーをあまりにもくつことはありません。

```
→ 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
```

ジオコーディングメカニズムをモデルにします。このでは、アドレスが `value` にされています。レコードがされ、がするにのみジオコーディングをするようにします。

```
# 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
```

デフォルトでは、ジオコーダーはlookupサービスとしてgoogleをします。それはやのようないがたくさんあります。は、 [ジオコードREADME](#) をしてください。

プロフィールでGoogleマップのアドレスをする

プロフィールで、リストのユーザーまたはグループのプロフィールフィールドと、Googleマップのアドレスフィールドをします。

```
- # app/views/profiles/show.html.haml
```

```
%h1 Contact Information
.profile_fields
  = render @profile_fields
.google_map{data: address_fields: @address_fields.to_json }
```

な@profile_fieldsと@address_fieldsがコントローラにされます

```
# 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
```

をし、マーカーをし、ズームやそののをJavaScriptでします。

The screenshot shows a web application interface with a navigation bar at the top containing links for 'Info', 'Contact', 'Corporate Vita', 'Work & Study', and 'More'. The 'Contact' tab is active. Below the navigation bar is a section titled 'Contact Information'. This section contains a list of contact details:

- Email: 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

To the right of the contact information is a Google Map. The map shows a red location pin over London, with labels for 'London', 'Paris', and 'Frankreich'. The map interface includes 'Karte' and 'Satellit' tabs, and a copyright notice for '© 2016 Google, ORIO'.

マップのマーカーをjavascriptでする

.google_map divがあり、マップになり、 dataとしてマーカーとしてするアドレスフィールドがあ

るとします。

えば

```
<!-- 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>
```

にするには `$(document).ready` とイベント `turbolinks` をで `turbolinks` イベントをすることなく、`jquery.turbolinks` のを。

で、たとえばフィルタリングやウィンドウなど、マップでのをするは、`コーヒースク립トクラス` でマップをすとです。

```
# 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
```

デフォルトでをついくつかのコーヒースク립トファイルをする、すべてのコーヒースク립トファイルでされるグローバル `App` をすとです。

に、によってはの `.google_map` ループし、それぞれの `App.GoogleMap` クラスのインスタンスを1つします。

```
# 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
```

コーヒースク립トクラスをしてをします。

`App.GoogleMap` `コーヒースク립トクラス` をすると、Google マップはのようになります

```
# 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
}

```

map_configuration オプションについては、googleの[MapOptions ドキュメント](#)と[コントロールの
にするガイド](#)をご覧ください。

までに、[ここでは](#) `google.maps.Map` クラスについて詳しくしています。

コーヒースクリプトクラスをしてマーカーをする

App.GoogleMap [コーヒースクリプトクラス](#)とマーカーが `.google_map div` の `data-address-fields` にされている `data-address-fields`、マップマーカーはのようにマップでできます。

```

# 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

```

マーカーオプションについては、Googleの[MarkerOptions](#)ドキュメントとマーカーガイドをご覧ください。

コーヒースクリプトクラスをしてをズームする

App.GoogleMap [コーヒースクリプトクラス](#)とgoogle.maps.Mapとして@mapとgoogle.maps.Marker Sとして@markers、マップズーム、すなわち、すべてのマーカーはのようになれるようにすることができます。まずマップでこのように

```

# 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

```

についてしくは、Google [LatLngBounds](#)のドキュメントをご覧ください。

モデルプロパティをjsonとしてする

アドレス・プロフィール・フィールドをマーカーとしてGoogleマップにするには、アドレス・フィールド・オブジェクトをjsonオブジェクトとしてjavascriptにすがあります。

のデータベース

[ApplicationRecord](#)オブジェクトにしてto_jsonをひすと、データベースのがにされます。

label、value、longitude、およびlatitudeをつProfileFields::Addressモデルがえられたvalue、

address_field.as_jsonはHash を address_field.as_json、

```
address_field.as_json # =>
  {label: "Work address", value: "Willy-Brandt-Straße 1\n10557 Berlin",
   longitude: ..., latitude: ...}
```

to_jsonによってjsonにされます。

```
address_field.to_json # =>
  "{\"label\":\"Work address\",\"value\":\"Willy-Brandt-Straße 1\\n10557 Berlin\\\", \"longitude\":..., \"latitude\":...}\"
```

これは、でjavascriptでlabelとvalueをできるようにするためにですたとえば、マップマーカのツールヒントをするなど。

そのの

のは、 as_jsonメソッドをオーバーライドすることでできます。

たとえば、 titleをするには、マージされたas_jsonハッシュにそのをめます。

```
# 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
```

のでは、 superをしてのas_jsonメソッドをびします。このメソッドは、オブジェクトのハッシュをし、なハッシュとマージします。

as_jsonとto_jsonいをするには、 as_json [ブログをご覧ください](#)。

ポジション

マーカをレンダリングするには、デフォルトでgoogle maps apiに、とがそれぞれlngとlatとしてされたpositionハッシュがです。

このハッシュは、アドレスフィールドのjsonをするときに、 javascript、 later、またはここででき

ます。

この`position`をアドレスフィールドの`json`として`as_json`するには、モデルの`as_json`メソッドをオーバーライドします。

```
# 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
```

オンラインでGoogleMapsとRailsのをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2828/googlemaps>とrailsの

17: HerokuにRailsアプリケーションをデプロイする

Examples

アプリケーションのデプロイ

Railsアプリケーションをむディレクトリにいることをし、Herokuでアプリケーションをします。

```
$ heroku create example
Creating [] example... done
https://example.herokuapp.com/ | https://git.heroku.com/example.git
```

このURL、<http://example.herokuapp.com>は、アプリができます。2のURL、[git@heroku.com example.git](https://git.heroku.com/example.git)は、リモートのgitリポジトリのURLです。

このコマンドは、されたgitリポジトリでのみしてください。heroku createコマンドは、このURLをす"heroku"というのgitリモートをにします。

app name "example"はオプションです。アプリをししないと、ランダムながされます。Herokuのアプリはグローバルなにあるので、「ブログ」や「wiki」などのながにされていることができます。デフォルトのからめ、でアプリケーションのをするほうがです。

に、コードをします。

```
$ 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
remote:      /tmp/build_49ba6c877f5502cd4029406e981f90b4/public/assets/application-
remote:      1bf5315c71171ad5f9cbef00193d56b7e45263ddc64caf676ce988cfbb6570bd.js
remote:      I, [2016-07-08T17:08:57.046951 #1222] INFO -- : Writing
remote:      /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
```

アプリケーションでデータベースをしているは、のコマンドをしてデータベースをでするがあります。

```
$ heroku run rake db:migrate
```

heroku run のコマンドは、Heroku heroku run されます。 のコマンドをすると、シェルセッションをできます。

```
$ heroku run bash
```

Web プロセスタイプをしている dyno が 1 つあることをします。

```
$ heroku ps:scale web=1
```

heroku ps コマンドは、アプリケーションののダイノスをしします。

```
$ 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)
```

あなたは heroku open 、ブラウザでアプリを heroku open することができ heroku open 。

```
$ heroku open
```

Herokuはherokuapp.comドメインのデフォルトのWeb URLをherokuapp.comます。にスケールアップするができたなら、のカスタムドメインをできます。

Herokuのプロダクションとステージングの

すべてのHerokuアプリケーションは、なくとも2つのでしますHerokuたちはそのプロダクションとぶとあなたのローカルマシンです。のがアプリケーションをしているは、のマシンごとに1つ、は1つがされています。、はテストをするためのテストもえています。ながら、このアプローチは、があまりていなくなるにつれてされます。たとえば、WindowsとMacはとも、HerokuのLinuxスタックとはなるをします。したがって、ローカルでするコードは、にするときとじですることはいつもではありません。

は、なりにしたステージングをつことです。これは、ステージングアプリケーションをホストする2のHerokuアプリケーションをすることでできます。ステージングでは、のユーザーにをえるに、プロダクションのようなでコードをチェックすることができます。

からめる

ローカルマシンでアプリケーションをしていて、それをHerokuにプッシュするができています。リモート、ステージング、およびプロダクションのをするがあります。にステージングにむというをるために、まずこれからめます

```
$ 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
```

デフォルトでは、heroku CLIはheroku git remoteをしてプロジェクトをします。ここでは、--remoteフラグでのをしているので、Herokuにコードをプッシュして、アプリケーションにしてコマンドをすると、のgit push herokuマスターとはしってえます。

```
$ git push staging master
...
$ heroku ps --remote staging
=== web: `bundle exec puma -C config/puma.rb`
web.1: up for 21s
```

ステージングアプリがしてしくしたら、プロダクションアプリをできます

```
$ 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
```

それで、2つの々のHerokuアプリケーションはステージングと1つのプロダクションがじょうにされているのと同じコードベースをできます。でどのアプリケーションをするのかをすることがあることを覚えておいてください。フラグ '--remote' をつか、git config をってデフォルトのアプリケーションをすることができます。

オンラインでHerokuにRailsアプリケーションをデプロイするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4485/heroku>にrailsアプリケーションをデプロイする

18: I18n -

- I18n.t "キー"
- I18n.translate "key" I18n.t("key") する I18n.t("key")
- I18n.t「キー」、カウント4
- I18n.t "key"、param1 "Something"、param2 "Else"
- I18n.t "doesnt_exist"、デフォルト "key"キーがないのデフォルトをする
- I18n.locale=>en
- I18n.locale =en
- I18n.default_locale=>en
- I18n.default_locale =en
- t " I18n.t("key") " I18n.t("key") とじですが、ひかれたアクション/テンプレートのスコープ

Examples

ビューでI18nをする

このYAMLロケールファイルがあるとします

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

あなたのタイトルをしたいは、これをうことができます

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

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

きのI18n

あなたは、にパラメータをすことができます_t

```
# 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) %>
```

そしてのをる

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

あなたは `I18n` があなたのためにをえるようにすることができます。に `count` をいます。

ロケールファイルをのようにするがあります

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

に、 `count` を `I18n.t` ヘルパーにして、したばかりのキーをします。

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

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

リクエストによるロケールの

ほとんどの、 `I18n` ロケールをすることができます。のセッション、のユーザ、またはURLパラメータについてロケールをすることができます。コントローラのいずれかに `before_action` をするか、 `ApplicationController` ですべてのコントローラに `before_action` をするとにできます。

```
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
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
```

URL ベース

`locale` パラメータはのような URL からあります

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

または

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

をするには、`routes` をして `scope` することができます。

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

これをうことで、`http://yourapplication.com/en/products` :en
`http://yourapplication.com/en/products` にアクセスすると、あなたのロケールは :en されます。わり
に `http://yourapplication.com/fr/products` にアクセスすると、:fr され :fr。しているときさらに、
あなたは、ルーティングエラーをすることはできません :locale PARAM をして、
`http://yourapplication.com/products` デフォルトの **!18n** ロケールをロードします。

セッションベースまたはパーシスタンスベース

これは、ユーザがをするためにボタン/フラグをクリックできることをとしています。このアクションは、セッションをのにするコントローラにルーティングできますユーザがされているは、
にデータベースにします

```
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
    !18n.locale = sanitize_language_param
    set_session_and_redirect
  end
end
```

なのリストをして `sanitize_language_param` をし、がしないのエラーをするがあります

。

がごくわずかなは、わりにのようにするがあります。

```
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
```

`change_language` アクションにいくつかのルートをすることをれないでください

デフォルトロケール

アプリケーションのデフォルトのロケールをするがあることにしてください。

`config/application.rb` するか、

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

または `config/initializers` フォルダにイニシャライザをします。

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

HTTP リクエストからロケールをする

IP についてアプリケーションロケールをするとながあります。これは `Geocoder` をしてにできます。

`Geocoder` がうくののでも、 `request location` をすることができます。

まず、 `Geocoder` を `Gemfile` します

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

Geocoderは、の `Rack::Request` オブジェクトに `location` および `safe_location` メソッドをするので、IP アドレスによるHTTPリクエストのをにできます。 `ApplicationController` `before_action` でこのメソッドをできます

```
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
```

0.0.0.0やlocalhostのようなものはななインターネットIPアドレスなので、これは `development` や `test` ではしないことにしてください。

と

Geocoderはにでありますが、ジオコーディングサービスをするようにするがありますは [こちら](#) を。そのくはにをけています。すべてのリクエストにしてサービスをびすと、パフォーマンスにをえるがあります。

これらにするには、のこともするがあります。

1. オフラインソリューション

GeoIP [ここ](#)をのようなをすると、ローカルデータファイルにしてをうことができます。これらのデータファイルをのにつがあるため、のでトレードオフがあります。

2. CloudFlare をする

CloudFlareをしてされるページには、コードがヘッダ `HTTP_CF_IPCOUNTRY` にされ、にジオコードされるオプションがあります。は [こちら](#)を [ご](#) [く](#) [だ](#) [さい](#)。

ActiveRecordモデルの

globalize gemは、 `ActiveRecord` モデルにをするのになソリューションです。これを `Gemfile` してインストールすることができます

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

Rails 5 をしているは、 `activemodel-serializers-xml` もするがあります

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

モデルをすると、モデルのをできます。たとえば、のようになります。

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

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

I18n.locale = :he
post.title # => גלובליזציה היא הטובה!
```

がなモデルをしたら、マイグレーションによってテーブルをするがあります。 `globalize`は `create_translation_table!` し `create_translation_table!` と `drop_translation_table!` 。

このでは、 `down` に `up` があり、 `change` するはありません。また、こののをにするには、にしたように、 `drop_translation_table!` を `create_translation_table!` にするがあります。の `Post` モデルのなはのとおりです。

```
class CreatePostsTranslationTable < ActiveRecord::Migration
  def up
    Post.create_translation_table! title: :string, text: :text
  end

  def down
    Post.drop_translation_table!
  end
end
```

のオプションのオプションをすこともできます

```
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
```

なカラムににのデータがあるは、をしてにテーブルにすることができます。

```
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

```

すべてのデータがにされたら、テーブルからされたをしてください。データにテーブルからされたをにするには、`remove_source_columns`オプションをにします。

```

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

  def down
    Post.drop_translation_table! migrate_data: true
  end
end

```

にしたテーブルにしいフィールドをすることもできます

```

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

```

HTML タグとでI18nをする

```

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

```

の`title`にの`_html`がされていることにしてください。

ビューでは、

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

オンラインでI18n - をむ <https://riptutorial.com/ja/ruby-on-rails/topic/2772/i18n---->

19: Prawn PDF

Examples

な

これは、をいたなアプローチです

```
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
```

はコードの `FundsPdf.new(@fund, view_context)` ます。 `FundsPdf` でヘルパーメソッドをうために、 `@fund` インスタンスと `view_context` で `FundsPdf` クラスをします。 `FundsPdf` はこのようにえる

```
class FundPdf < Prawn::Document

  def initialize(fund, view)
    super()
    @fund = fund
    @view = view
    upper_half
    lower_half
  end

  def upper_half
    logopath = "#{Rails.root}/app/assets/images/logo.png"
    image logopath, :width => 197, :height => 91
    move_down 10
    draw_text "Receipt", :at => [220, 575], size: 22
    move_down 80
    text "Hello #{@invoice.customer.profile.first_name.capitalize},"
  end

  def thanks_message
    move_down 15
    text "Thank you for your order.Print this receipt as
    confirmation of your order.",

```

```
      :indent_paragraphs => 40, :size => 13
    end
  end
end
```

これは、Prawn gemをしてクラスをしてPDFをするのの1つです。

な

あなたはGemとPDF MIMEをするがありますmime_types.rbのにしてください。たちはPDF MIMEタイプについてルールにするがあります。

その、たちはPrawnを使ってPdfをなですることができます

これはなりてです

```
pdf = Prawn::Document.new
pdf.text "Hello World"
pdf.render_file "assignment.pdf"
```

たちはのブロックでそれをうことができます

```
Prawn::Document.generate("implicit.pdf") do
  text "Hello World"
end
```

ブロックで

```
Prawn::Document.generate("explicit.pdf") do |pdf|
  pdf.text "Hello World"
end
```

オンラインでPrawn PDFをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4163/prawn-pdf>

20: Rails 5

Examples

Ruby on Rails 5 APIの

しいRails 5 APIをするには、をいてのコマンドをします。

```
rails new app_name --api
```

のファイルがされます。

```
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/inflexions.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
```

このファイルは、`app_name`というしいフォルダにされます。プロジェクトをするためになすべてのアセットとコードがまれています。

フォルダーをし、をインストールします。

```
cd app_name
bundle install
```

また、データベースをするがあります。RailsはSQLiteをデフォルトのデータベースとしてします。するには、のコマンドをします。

```
rake db:setup
```

すぐあなたのアプリケーションをしてください

```
$ rails server
```

`http://localhost:3000`でブラウザをくと、っているしいのAPIがしているはずです。

Ruby on Rails 5をRVMにインストールする

RVMはあなたのルビーバージョンをし、あなたのをするらしいツールです。

すでにRVMがインストールされているとして、これらのでなバージョンのrubyをするには、をいでのコマンドをします。

```
$ rvm get stable
$ rvm install ruby --latest
```

をしてあなたのルビーバージョンをしてください

```
$ ruby -v
> ruby 2.3.0p0
```

Rails 5をインストールするには、まず、のRubyバージョンをしてしいgemsetをし、にレールをインストールします。

```
$ rvm use ruby-2.3.0@my_app --create
$ gem install rails
```

レールのバージョンをするには、のコマンドをします。

```
$ rails -v
> Rails 5.0.0
```

オンラインでRails 5をむ <https://riptutorial.com/ja/ruby-on-rails/topic/3019/rails-5>

21: Rails 5 APIのオートメーション

Examples

Railsによるauthenticate_with_http_token

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

このエンドポイントを`curl`テストするには、のようなリクエストをいます。

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

オンラインでRails 5 APIのオートメーションをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7852/rails-5-apiのオートメーション>

22: Rails API

Examples

APIアプリケーションの

APIサーバとなるRailsアプリケーションをするには、よりされたRails in Rails 5のサブセットから始めることができます。

しいRails APIアプリケーションをするには

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

`--api`は、APIをするになをする事です。これには、セッション、クッキー、アセット、およびRailsをブラウザでさせるものがまれます。

また、しいリソースをするときにビュー、ヘルパー、アセットをしないようにジェネレータをします。

Web `ApplicationController`の`ApplicationController`とAPIアプリケーションをすると、Webバージョンは`ActionController::Base`からされていますが、APIバージョンは`ActionController::API`からされています。

オンラインでRails APIをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4305/rails-api>

23: Rails Cookbook - なレールレシピ/とコーディングテクニック

Examples

レールコンソールをしてテーブルでする

テーブルの

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

のテーブルをします。

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

のからインデックスをする

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

ここで、`country`は、マイグレーションファイルので、にすように、`users`テーブルにインデックスがにされています。

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

キーをする

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

そこでは、`menus has_many food_items`とそれぞれのがあります。

を

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

えば-

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

Rails メソッド - ブールをす

Railsモデルのどのメソッドもブールをすことができます。

な -

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

びブールをすなメソッド -

```
##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
```

だから、じメソッドは、かのわりにブールをします:)

エラーの - のメソッド `where` for

には、 ActiveRecord::Relation ではないされたレコードのコレクションにして where クエリをすること where ます。Heauseは、 Where が ActiveRecord ではなく Array ことをっています。

これは、 Joins をしてながあります。

-

id = 10のユーザUserではないアクティブなすべてのユーザプロフィールUserProfileをつけるがあるとします。

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

したがって、のクエリは、 map が where ではしない array をすので、 map にしmap。

しかし、をして、それをさせる、

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

joins は map ようなレコードをしmap が、 Array ではなく ActiveRecord なります。

オンラインで Rails Cookbook - なレールレシピ/とコーディングテクニックをむ

<https://riptutorial.com/ja/ruby-on-rails/topic/7259/rails-cookbook----なレールレシピ-とコーディングテクニック>

24: rails アプリケーションに Amazon RDS をする

き

なコネクタをインストールしてAWS RDSインスタンスをし、database.ymlファイルをする。

Examples

MYSQL RDSとあなたのレールアプリケーションをしているとします。

MYSQL データベースをする

1. amazonアカウントにログインし、RDSサービスをし
2. インスタンスタブから「Launch DB Instance」をクリックします。
3. MySQL Community Editionがされると、select ボタンをクリック select します。
4. データベースのをしします。例えば、production とクリックして next step をクリックします。
5. mysql version, storage size, DB Instance Identifier, Master Username and Password、 next step をクリックします
6. Database Name をし、[Launch DB Instance Database Name] をクリックします。
7. すべてのインスタンスがされるまでおちください。インスタンスがされると、エンドポイントをつけて、このエントリポイントをコピーしますこれはホストとばれます

コネクタのりけ

MySQLデータベースアダプタをプロジェクトのgemfileにし、

```
gem 'mysql2'
```

したをインストールし、

```
bundle install
```

のデータベースアダプタには、

- PostgreSQLのgem 'pg'
- gem 'activerecord-oracle_enhanced-adapter' for Oracle
- gem 'sql_server' for SQL Server

プロジェクトのdatabase.ymlファイルをするconfig / database.ymlファイルをく

```
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 post. For MYSQL 3306
```

オンラインでrailsアプリケーションにAmazon RDSをするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/10922/railsアプリケーションにamazon-rdsをする>

25: Rails アプリケーションのテスト

Examples

テスト

ユニットテストでは、アプリケーションのをでテストします。、テストのユニットはクラスまたはモジュールです。

```
let(:gift) { create :gift }

describe '#find' do
  subject { described_class.find(user, Time.zone.now.to_date) }
  it { is_expected.to eq gift }
end
```

ソース

このテストは、なりでなものであればです。

リクエストテスト

テストは、ユーザーのをするエンドツーエンドのテストです。

```
it 'allows the user to set their preferences' do
  check 'Ruby'
  click_on 'Save and Continue'
  expect(user.languages).to eq ['Ruby']
end
```

ソース

このテストでは、ユーザーのフローにをて、システムのすべてのレイヤーをして、JavaScript をレンダリングすることさえできます。

オンラインでRailsアプリケーションのテストをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7853/railsアプリケーションのテスト>

26: Railsエンジン - モジュールレール

き

Railsエンジンの

エンジンは、それをホストするアプリケーションにをするためにできるさなRailsアプリケーションです。Ruby on Railsアプリケーションをするクラスがある`Rails::Application`から、そののくをする`Rails::Engine`、エンジンをするクラス。のRailsアプリケーションは、よりくのをえたなるエンジンであるとえます。

- レールプラグイン`new my_module --mountable`

Examples

モジュラーアプリをする

まず、しいRuby on Railsアプリケーションをしましょう。

```
rails new ModularTodo
```

のステップは、エンジンをすることです

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

エンジンをするための「エンジン」フォルダもしますたとえそれがあったとしても。

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

エンジンは、のように、`gemspec`ファイルがしています。をけるためにいくつかののをれてみましょう。

```
#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
```

これで、TodoエンジンをアプリケーションのGemfileにするがあります。

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

`bundle install` しましょう。のリストにはがされます

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

らしいです、たちのTodoエンジンがしくロードされていますコーディングをするに、にTodoエンジンをマウントします。アプリのroutes.rbファイルでこれをうことができます。

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

/マウントしていますが、 /todoにアクセスできるようにすることもできます。モジュールは1つしかないなので、 /はありません。

すぐあなたのサーバーをし、あなたのブラウザですることができます。まだコントローラ/ビューをしていないので、デフォルトのRailsビューがされるはずです。すぐやろう

Todo リストの

たちはTodoモジュールのでTaskというのモデルをengine.rbしようとしていますが、アプリケーションからデータベースをしくするために、さなをengine.rbファイルにするがあります。

```
#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
```

```
        end
      end
    end

    end
  end
```

それで、アプリケーションからマイグレーションをすると、Todoエンジンのマイグレーションもロードされます。

Taskモデルをしましょう。 scaffoldコマンドはエンジンフォルダからするがあります。

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

フォルダからをします。

```
rake db:migrate
```

さて、Todoエンジンのでルートルートをするだけです。

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

あなたはそれでぶことができる、タスクをする、それらをする...ああって、がしていませんなぜ JQueryがみまれていないようですので、エンジンのapplication.js ファイルにしましょう

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

さて、たちはをすることができます

オンラインでRailsエンジン - モジュールルールをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9080/railsエンジン---モジュールルール>

27: Railsでのユーザ

き

Deviseはにパワフルなです。インストールにサインアップ、サインイン、サインアウトをすることができます。さらに、ユーザはアプリケーションにとをできます。ユーザがしたい、Deviseものっています。ユーザは、とにじて、サインアップとサインインフォームをカスタマイズすることもできます。あなたがめてのは、のログインをすることをおめします。

`rails generate devise:install`をしてdevise configsをするとき、deviseはでのをするようします。

すでに`USER`モデルがある、このコマンドをすると、`rails generate devise USER`はの`USER`モデルになをします。

このヘルパーメソッド`before_action :authenticate_user!`してください`before_action :authenticate_user!`あなたのコントローラーのに`user`がログインしているかどうかをチェックします。そうでないは、サインインページにリダイレクトされます。

Examples

Deviseをした

gemfileにgemをする

```
gem 'devise'
```

に、`bundle install`コマンドをします。

なファイルをするには、コマンド`$ rails generate devise:install`をします。

でDeviseメーラーのデフォルトURLオプションをするでのをします

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

あなたの`config/environments/development.rb`

に`config/environments/production.rb`この`config/environments/production.rb`ファイルをしてadd

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

`$ rails generate devise USER`ここで、`USER`はをするクラスです。

に、`rake db:migrate`を`rake db:migrate`と、すべてされます。

カスタムビュー

ビューをするがある、すべてのビューをアプリケーションにコピー `rails generate devise:views` ジェネレータをできます。その、にじてすることができます。

アプリケーションにのDeviseモデルUserやAdminなどがある、Deviseはすべてのモデルでじビューをしています。Deviseは、ビューをカスタマイズするなをします。

`config/initializers/devise.rb` ファイルで `config.scoped_views = true` を `config/initializers/devise.rb` ます。

ジェネレータをしてスコープきビューをすることもできます。 `rails rails generate devise:views users`

なモジュールやなモジュールのように、いくつかのビューセットしは、`-v` フラグをし `rails generate devise:views -v registrations confirmations`

デバイスコントローラフィルタとヘルパー

`devise` をしてユーザをうコントローラをするには、この `before_action` をしますあなたのデバイスモデルが「ユーザ」であるとします。

```
before_action :authenticate_user!
```

ユーザーがログインしているかどうかをするには、のヘルパーをします。

```
user_signed_in?
```

ログインしているユーザーのは、のヘルパーをします。

```
current_user
```

このスコープのセッションにアクセスできます。

```
user_session
```

- あなたのDeviseモデルが `User` ではなく `Member` とばれる、の `user` を `member`

オムニウス

まず、あなたの `auth` をし、それを `Gemfile` します。のリストはこちら <https://github.com/intridea/omniauth/wiki/List-of-Strategies>

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

あなたはルールミドルウェアにのようになすることができます

```
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
```

デフォルトでは、OmniAuthはルートに`/auth/:provider`をし、これらのパスをしてすることができます。

デフォルトでは、がした、omniauthは`/auth/failure`リダイレクトします

has_secure_password

ユーザーモデルの

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

ユーザーモデルに**has_secure_password**モジュールをする

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

パスワードでしいユーザーをできるようになりました

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

authenticateメソッドでパスワードをする

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

has_secure_token

ユーザーモデルの

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

しいユーザーをすると、トークンとauth_tokenがにされます

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

regenerate_tokenとregenerate_auth_tokenをしてトークンをできます

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

オンラインでRailsでのユーザをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1794/railsでのユーザ>

28: Railsでを

Examples

ルール101の

ステップ1しいRailsアプリケーションを

```
gem install rails -v 4.1
rails new angular_example
```

ステップ2ターボリンクのりし

ターボリンクをりすには、Gemfileからターボリンクをするがあります。

```
gem 'turbolinks'
```

app/assets/javascripts/application.jsからrequireをしapp/assets/javascripts/application.js。

```
//= require turbolinks
```

ステップ3AngularJSをアセットパイプラインにする

AngularがRailsのアセットパイプラインでするようにするには、Gemfileにするがあります

```
gem 'angular-rails-templates'
gem 'bower-rails'
```

コマンドを

```
bundle install
```

AngularJSのをインストールできるようにbowerしてください

```
rails g bower_rails:initialize json
```

bower.jsonにAngularを

```
{
  "name": "bower-rails generated dependencies",
  "dependencies": {
    "angular": "latest",
    "angular-resource": "latest",
    "bourbon": "latest",
    "angular-bootstrap": "latest",
    "angular-ui-router": "latest"
  }
}
```

`bower.json`がされたので、それらをインストールしてみましょう

```
bundle exec rake bower:install
```

ステップ4 アプリをする

`app/assets/javascript/angular-app/`のフォルダをします。

```
templates/
modules/
filters/
directives/
models/
services/
controllers/
```

`app/assets/javascripts/application.js`に `require`、**Angular**には `require`、テンプレートヘルパー、**Angular app**ファイルをします。このような

```
//= 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
```

ステップ5 アプリをブートストラップする

`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'
)
```

app/assets/javascripts/angular-app/modules/example.js.coffee.erb **Angularモジュールをし**
 app/assets/javascripts/angular-app/modules/example.js.coffee.erb。

```
@exampleApp = angular.module('app.exampleApp', [ # additional dependencies here ])
  .run(-> console.log 'exampleApp running' )
```

app/assets/javascripts/angular-app/controllers/exampleCtrl.js.coffeeにこのアプリのコントローラをします。

```
angular.module('app.exampleApp').controller("ExampleCtrl", [ '$scope', ($scope)->
  console.log 'ExampleCtrl running' $scope.exampleValue = "Hello angular and rails" ])
```

は、**Rails**にルートをして、**Angular**にをします。 config/routes.rb

```
Rails.application.routes.draw do get 'example' => 'example#index' end
```

そのルートにするために**Rails**コントローラをします

```
rails g controller Example
```

app/controllers/example_controller.rb

```
class ExampleController < ApplicationController
  def index
    end
end
```

ビューでは、どのアプリケーションとどのコントローラがこのページをするかをするがあります。したがって、 app/views/example/index.html.erbのapp/views/example/index.html.erb。

```
<div ng-app='app.exampleApp' ng-controller='ExampleCtrl'>
  <p>Value from ExampleCtrl:</p>
  <p>{{ exampleValue }}</p>
</div>
```

このアプリケーションをするには、**Rails**サーバーをし、 [http:// localhost3000 / example](http://localhost3000/example)にアクセスしてください。

オンラインで**Rails**でをするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3902/rails>でをする

29: Railsのアップグレード

Examples

Rails 4.2からRails 5.0へのアップグレード

Railsアプリケーションをアップグレードするに、**ずGit**などのバージョンシステムにコードをしてください。

Rails 4.2からRails 5.0にアップグレードするには、Ruby 2.2.2をしているがあります。にじてRubyのバージョンをアップグレードしたら、Gemfileにしてのをします

```
gem 'rails', '4.2.X'
```

に

```
gem 'rails', '~> 5.0.0'
```

コマンドラインでのコマンドをします。

```
$ bundle update
```

に、のコマンドをしてタスクをします。

```
$ rake rails:update
```

これはファイルをするのにちます。ファイルをきするかどうかをねるプロンプトがされ、いくつかのオプションがあります。

- Y - はい、き
- n - いいえ、きしない
- a - all、これとのすべてをきする
- q - する、する
- d - diff、いものとしいもののをいをする
- h - ヘルプ

、いファイルとしいファイルのをいをべて、ましくないがないことをするがあります。

Rails 5.0 ActiveRecordモデルは、 ActiveRecord::Baseではなく ApplicationRecordからしてい ApplicationRecord。 ApplicationRecordどのようにたすべてのモデルのスーパー、ある ApplicationControllerコントローラのスーパークラスです。モデルがされるこのしいをするには、 app/models/フォルダに application_record.rbというファイルをし、そのファイルのをのようにするがあります。

```
class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

Rails 5.0では、ややなるコールバックもされます。 `false` をすコールバックはコールバックチェーンをさせません。これは、Rails 4.2とはなり、ききコールバックがされることをします。アップグレードすると、Rails 4.2のはりますが、Rails 5.0のにりえるには、のものをします。

```
ActiveSupport.halt_callback_chains_on_return_false = false
```

`config/application.rb` ファイルに `config/application.rb` ます。コールバックチェーンをにさせるには、 `throw(:abort)` びします。

Rails 5.0では、 `ActiveJob` はRails 4.2のように `ActiveJob::Base` ではなく `ApplicationJob` からし `ApplicationJob` 。 Rails 5.0にアップグレードするには、 `app/jobs/` フォルダに `application_job.rb` というファイルをしします。そのファイルのをののようにしします。

```
class ApplicationJob < ActiveJob::Base
end
```

に、 `ActiveJob::Base` ではなく `ApplicationJob` からするすべてのジョブをするがあります。

Rails 5.0のそののもきなのは、コードをするはありませんが、Railsアプリケーションでコマンドラインをするがわかります。 `bin/rails`、または `rails` だけでタスクやテストをすることができます。たとえば、 `$ rake db:migrate` をするわりに、 `$ rails db:migrate` できます。 `$ bin/rails` をすると、なすべてのコマンドをできます。 `bin/rails` できるタスクのくはまだ `rake` をってしていることにしてください。

オンラインでRailsのアップグレードをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3496/railsのアップグレード>

30: Railsはコマンドをする

き

`rails generate GENERATOR_NAME [args] [options]`。

`rails generate`をして、なジエネレータをリストします。エイリアス `rails g`。

パラメーター

パラメータ	
<code>-h / --help</code>	ジエネレータコマンドにするヘルプをする
<code>-p / --pretend</code>	Pretend Modeジエネレータをしますが、ファイルをまたはしません。
<code>field:type</code>	'field-name'はするので、'type'はのデータです。 <code>field:type 'type'</code> にできるは、にあります。

`field:type 'type'`にできるはのとおりです。

データ・タイプ	
<code>:string</code>	さなテキストのはが255です
<code>:text</code>	のようないテキストの
<code>:binary</code>	、オーディオ、ビデオをむデータの
<code>:boolean</code>	の
<code>:date</code>	のみ
<code>:time</code>	だけ
<code>:datetime</code>	
<code>:float</code>	のないフロートの
<code>:decimal</code>	をでする
<code>:integer</code>	の

Examples

Railsはモデルをする

モデルになdbマイグレーションとボイラプレート・テスト・ファイルをにするActiveRecordモデルをするには、このコマンドをします

```
rails generate model NAME column_name:column_type
```

'NAME'はモデルのです。'field'はDBテーブルのので、'type'はののです name:stringまたはbody:text。サポートされているののについては、「」をしてください。

キーをするには、 belongs_to:model_name します。

だから、 usernameとemailち、 SchoolにしているUserモデルをセットアップしたければ、のようにします

```
rails generate model User username:string email:string school:belongs_to
```

rails gは、 rails generateためのです。これによりじがられます

```
rails g model User username:string email:string school:belongs_to
```

Railsによるの

のコマンドをして、からルールファイルをできます。

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

このコマンドでサポートされているすべてのオプションのリストについては、 rails generate migrationようになしでコマンドをできます。

たとえば、 first_nameフィールドとlast_nameフィールドをusersテーブルにするは、のようにします。

```
rails generate migration AddNamesToUsers last_name:string first_name:string
```

Railsはのファイルをします。

```
class AddNamesToUsers < ActiveRecord::Migration[5.0]
  def change
    add_column :users, :last_name, :string
    add_column :users, :first_name, :string
  end
end
```

に、でのコマンドをして、のをデータベースにします。

5.0

```
rake db:migrate
```

5.0

```
rails db:migrate
```

をさらになくするために、`generate`を`g`きえることができます。

レールはをする

にの**CRUD**アプリケーション/テストでないり、はされません。これはあなたのWebアプリケーションでなくのファイルビュー/モデル/コントローラをし、い:(をきこすがあります。

モデル、コントローラ、ビュー、アセット、およびテストをむしいオブジェクトのなをするには

、`rails g scaffold`コマンドをし`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
```

に、`rake db:migrate`をしてデータベーステーブルをします。

に、[http://localhost3000 / widgets](http://localhost3000/widgets)にアクセスすると、にな**CRUD**スカフールドがされます。

31: Rails ベストプラクティス

Examples

りさないでくださいドライ

きれいなコードをするために、RailsはDRYのにいます。

なり、なりくのコードをします。のでじコードをするのではなく、なコードをします。これにより、エラーがし、コードがきれいにたれ、コードをしてからするというがされます。また、じコードののをするよりも、あるでコードをするがです。したがって、コードをモジュールしてにします。

あなたのモデルにコードをいているので、また、コントローラーでコードをいているだけなので、Fat Model、Skinny ControllerはDRYです。

```
# 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
```

これはまた、メソッドがされ、APIのやりでパラメータをすことによってがされるAPIにつながります。

コンベンションオーバーコンフィグレーション

Railsでは、データベースのコントローラ、ビュー、モデルをています。

のをらすために、Railsはルールをしてアプリケーションのをにします。のルールをしてもいませんが、はRailsがするにうことをおめします。

これらのは、をスピードアップし、コードをかつみやすくして、アプリケーションでのなナビゲーションをにします。

ではのもくなります。がるがないだけでなく、でをけることができるように、Railsにはたくさん

のがあります。なぜすべてがしいのかをらずにすばらしいアプリケーションをすることはです。

えば

キーidつordersというデータベーステーブルがある、するモデルはorderとばれ、すべてのロジックをするコントローラはorders_controllerというにorders_controllerます。ビューはなるアクションでされます。コントローラーにnewアクションとeditアクションがあるは、newビューとeditビューもあります。

えば

アプリケーションをするには、にrails new app_nameしrails new app_name。これにより、Railsアプリケーションのインフラとをする70のファイルとフォルダがされます。

それはをむ

- モデルデータベース、コントローラ、およびビューをするフォルダ
- アプリケーションのテストをするフォルダ
- JavascriptやCSSファイルなどのWebアセットをするフォルダ
- HTTP 400レスポンスのデフォルトファイルファイルが見つからない
- のく

モデル、スキニーコントローラ

「モデル、スキニーコントローラ」は、MVCのMとCがににくをします。つまり、レスポンスにのないうロジックは、にはテストなでモデルにるべきです。、「スキニー」コントローラは、ビューとモデルののなるらしいインタフェースです。

には、これにはさまざまなタイプのリファクタリングがですが、すべてが1つのアイデアになります。コントローラではなく、モデルのではないロジックをすることで、をするだけでなくであれば、のコンテキストでコードをテストすることにもしました。

なをてみましょう。のようなコードがあるとします。

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

のようにすることができます

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

に、ロジックをポストモデルにすると、のようになります。

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

default_scopeにしてください

ActiveRecordにはdefault_scopeまれており、デフォルトでモデルをにスコープします。

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

のコードは、モデルのクエリをするとにされているをします。

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

そのスコープは、にえるものの、のされたがあり、あなたがまないかもしれません。

default_scopeとorder

default_scopeでorderをしたので、Postびしorderはデフォルトをきするのではなく、のオーダーとしてされます。

```
Post.order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."created_at"
DESC, "posts"."updated_at" DESC
```

これはおそらくあなたがむではありません。にスコープからorderをすることでこれをにすることができます

```
Post.except(:order).order(updated_at: :desc)
```

```
SELECT "posts".* FROM "posts" WHERE "posts"."published" = 't' ORDER BY "posts"."updated_at"
DESC
```

default_scopeとモデルの

のActiveRecord::Relationに、default_scopeは、されたモデルのデフォルトをします。

のでは、Postはデフォルトでwhere(published: true)されているので、Postしいモデルにもされています。

```
Post.new # => <Post published: true>
```

unscoped

`default_scope`は`unscoped`にびすことでクリアすることができますが、これにはもあります。たとえば、STIモデルをえてみましょう。

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

デフォルトでは、`Post`にするは`Scope`され、`'Post'`をむが`type`ます。しかし、`unscoped`はこの`default_scope`とにこれをクリアするので、`unscoped`をするはこれにもれるがあります。

```
Post.unscoped.where(type: 'Post').order(updated_at: :desc)
```

unscoped およびモデル

PostとUser

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

class User < ApplicationRecord
  has_many :posts
end
```

々の`User`すると、するをることができます。

```
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]]
```

しかし、あなたは`posts`から`default_scope`をクリアしたいので、`unscoped`をう

```
user.posts.unscoped
```

```
SELECT "posts".* FROM "posts"
```

これにより、`default_scope`とに`user_id`がされ`user_id`。

default_scope

そのすべてにかかわらず、`default_scope`をすることがながあります。

のサブドメインがアプリケーションからされるが、されたデータがあるマルチテナントシステムをえてみましょう。このをする1つのは`default_scope`です。ののはここではになります。

```
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
```

あなたがするがあるのは、リクエストのいで`Tenant.current_id`をすることだけです。`tenant_id`をむテーブルは、のコードなしでにスコープになります。レコードをインスタンスすると、されたテナントIDがにされます。

このユースケースのなは、スコープがごとに1され、されないことです。ここでスコープを`unscoped`があるのケースは、スコープでされるバックグラウンドワーカーなどのなケースです。

あなたはそれをとしない**YAGNI**

あなたがについて「YAGNI」あなたはそれをとしないだろうとすることができるなら、あなたはそれをしないほうがよいでしょう。シンプルさにをてることで、くのをできます。とにかくこのよなをすると、がするがあります。

Overengineering

がそれよりもであれば、それはされています。、これらの「まだされていない」は、されたでされることはなく、されるとリファクタリングするがあります。の、にパフォーマンスのは、しばしばのにつながり、にっていることがします。

コードブロッティング

Code Bloatはななコードをします。これは、えは、、、またはパターンのったによってじる。コードベースは、しにくく、し、するののがかかります。

フィーチャークリープ

フィーチャークリープとは、のコアをえるしいをし、になにつながることをします。

い

なをするためにできるは、なをするためにやされます。のにかかります。

ソリューション

キス。それは、かなままに

KISSによると、シンプルにされていれば、ほとんどのシステムがです。シンプルシティは、さをするためのなです。これは、えは、「」にうことによつてすることができます。

YAGNI - あなたはそれをとしない

ないほうがいいですね。すべてのをえてください。にですかもしあなたがYAGNIであるとうなら、それをれてください。それがなときにするがいです。

リファクタリング

はにされています。リファクタリングでは、がベストプラクティスにってされていることをし、パッチにしないようにすることができます。

ドメインオブジェクトファットモデルなし

「Fat Model、Skinny Controller」はにいますが、コードベースがしめるとスケールががらなくなります。

モデルのについてえてみましょう。モデルののはですかそれはビジネスロジックをするのですかそれはのをするのですか

いいえ、そのは、パーシスタンスとそのをすることです。

ビジネスロジックは、レスポンスにしないロジックやパーシスタンスのロジックとに、ドメインオブジェクトにめるがあります。

ドメインオブジェクトは、のドメインで1つのしかたないようにされたクラスです。あなたのクラスで、らがしたを「[しみの](#)」にしましょう。

には、あなたはスキニーのモデル、スキニーのビューとスキニーコントローラにあってするがあります。ソリューションのアーキテクチャは、しているフレームワークのをけてはいけません。

えは

ストライプでおに15のをするマーケットプレイスだとします。15をすると、にじてがわることをします。ストライプは2.9+ 30¢です。

としておいいただく $\text{amount} * 0.15 - (\text{amount} * 0.029 + 0.30)$ です。

このロジックをモデルにきまないでください

```
# 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
```

しいいとするとすぐに、このモデルでこのをすることはできません。

また、よりくのビジネスロジックをしめると、`Order`オブジェクトはをいめます。

なのからにされたので、ドメインオブジェクトをむ

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

# lib/commission.rb
class Commission
  SERVICE_COMMISSION = 0.15

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

  private

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

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

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

```
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
```

ドメインオブジェクトをすると、のようなアーキテクチャのがあります。

- ロジックをしてオブジェクトをインスタンスするためのファクトリはないため、テストはにです。
- メッセージ`amount`をけるすべてのもののでします。
- それぞれのドメインオブジェクトをさくち、なをし、よりいをしします。
- **ではなく、によるしいでにできます。**
- Ruby on Railsアプリケーションでにする`User`オブジェクトをつをなくします。

はに`lib`にドメインオブジェクトをくのがきです。そのは、`autoload_paths`にすることをれないでください

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

また、コマンド/クエリのパターンにって、よりのドメインオブジェクトをすることもできます。そのような、これらのオブジェクトを`app/commands`にれるのは、すべての`app`サブディレクトリがロードパスににされるため、よりいになるがあります。

オンラインでRailsベストプラクティスをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1207/rails-best-practices>

33: RSpecとRuby on Rails

RSpecは、Rubyのテストフレームワークであり、のドキュメントでされているように、*RSpec*はRubyプログラマーのツールです。

このトピックでは、Ruby on RailsでRSpecをするについてします。RSpecのについては、RSpecのトピックをしてください。

Examples

RSpecのインストール

RSpecをRailsプロジェクトにするは、`rspec-rails` gemをするがあります。これはヘルパーやスベックファイルをにします例えば、モデル、リソース、またはscaffoldを`rails generate`をもってするなど。

`rspec-rails`の`:development`と`:test`のグループに`rspec-rails`をしGemfile。

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

`bundle`をしてをインストールします。

それをする

```
rails generate rspec:install
```

これにより、のファイルとともに、テストの`spec/`フォルダがされます。

- `.rspec`にはコマンドライン`rspec`ツールのデフォルトオプションがまれています
- `spec/spec_helper.rb`は、RSpecオプションがまれています。
- `spec/rails_helper.rb`は、RSpecとRailsをにするためのよりなオプションをします。

これらすべてのファイルは、あなたがいめるためになデフォルトでかれています。テストスイートがするにつれて、をしてニーズにわけてをすることができます。

オンラインでRSpecとRuby on Railsをむ <https://riptutorial.com/ja/ruby-on-rails/topic/5335/rspecとruby-on-rails>

34: Ruby on Railsのコードとクリーンアップのためのツール

き

なであっても、なRailsアプリケーションをしながらコードをきれいにすることは、にしいです。いにも、このをはるかににするのカテゴリがあります。

Examples

コードをしやすく、にしたいは、コードのとクリーンアップについていくつかのをてください。

これはにのをきばした。きのは、 $N + 1$ のクエリだけでなく、ににみまれたをすべてすのにちます。インストールしてのさまざまなルートをとると、するのあるデータベースクエリをすメッセージがされます。これはすぐにでき、アプリケーションののにちます。

Rails ベストプラクティス

Railsのコードのいをつけるコードアナライザ。それはさまざまなをします。スコープアクセスをしたり、ルートをしたり、データベースインデックスをしたりすることはできますが、コードをしてベストプラクティスをぶについてのよりいをるためのらしいがたくさんあります。

ルーコップ

あなたのコードがRubyコミュニティコードのガイドラインにしているかどうかをするためにできるRubyスタティックコードアナライザ。これはコマンドラインをしてスタイルをします。なりて、におけるObjectto_sの、またはのメソッドなど、なコードリファクタリンググッズがたくさんあります。

いことは、Rubyスタイルガイド100にっていないとアナライザーがにになるがあるつまり、にがたくさんあるか、またはしていなくてもをでむなど。

それは、4つのサブアナライザーとばれるスタイル、くず、メトリクス、およびルールにかれています。

オンラインでRuby on Railsのコードとクリーンアップのためのツールをむ

<https://riptutorial.com/ja/ruby-on-rails/topic/8713/ruby-on-railsのコードとクリーンアップのためのツール>

35: Ruby on Railsのネストされたフォーム

Examples

Ruby on Railsでネストされたフォームをする

にっているもののモデルと`has_many`リレーションをむモデル。

```
class Project < ApplicationRecord
  has_many :todos
end

class Todo < ApplicationRecord
  belongs_to :project
end
```

ProjectsController

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

ネストされたフォームでは、オブジェクトをつオブジェクトをにできます。

```
<%= 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 %>
```

@projectがProject.new @projectをして、しいProjectオブジェクトをするためのもの、Todoオブジェクトをするためのものと同じようにするがあるため、これをうはいくつかあります。

1. Projectscontrollerでは、newメソッドで、@todo = @project.todos.buildまたは@todo = @project.todos.newとくことで、しいTodoオブジェクトをインスタンスできます。
2. ビューでもこれをうことができます <%= f.fields_for :todos, @project.todos.build %>

なパラメータの、のようにパラメータをめることができます。

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

```
end
```

`Project` オブジェクトのによって `Todo` オブジェクトがされるため、 `Project` モデルでのをしてこのことをするがあります。

```
accepts_nested_attributes_for :todos
```

オンラインでRuby on Railsのネストされたフォームをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8203/ruby-on-rails>のネストされたフォーム

36: アクティブジョブ

き

アクティブジョブは、ジョブをし、それらをさまざまなキューバックエンドでするためのフレームワークです。これらのジョブは、にされているクリーンアップから、までのすべてになります。さなにかくりけて、してできるもの

Examples

ジョブをする

```
class GuestsCleanupJob < ApplicationJob
  queue_as :default

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

ジョブをエンキューする

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

オンラインでアクティブジョブをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8996/アクティブジョブ>

37: アクティブなジョブ

Examples

ま

Rails 4.2、Active Jobはジョブをし、さまざまなキューバックエンドでできるようにするためのフレームワークです。ブロックされておらず、してできるまたはのタスクは、アクティブなジョブのいです。

サンプルジョブ

```
class UserUnsubscribeJob < ApplicationJob
  queue_as :default

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

ジェネレータをしたアクティブジョブの

```
$ rails g job user_unsubscribe
```

オンラインでアクティブなジョブをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8033/アクティブなジョブ>

38: アクティブモデルシリアライザ

き

ActiveModelSerializers、つまりAMSは、JSONのに 'をえたconvention over configuration'をもたらします。ActiveModelSerializersは、シリアライザとアダプタの2つのコンポーネントでします。シリアライザは、どのとりレーションシップをシリアルするがあるかをします。アダプターは、とりレーションシップをどのようにシリアライズするかをします。

Examples

シリアライザの

```
class SomeSerializer < ActiveModel::Serializer
  attribute :title, key: :name
  attributes :body
end
```

オンラインでアクティブモデルシリアライザをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9000/アクティブモデルシリアライザ>

39: アクティブレコード

Examples

する

`belongs_to` アソシエーションは、のモデルと11のをするため、するモデルのインスタスは、のモデルの1つのインスタスにします。

たとえば、アプリケーションにユーザーとがまわっていて、をに1のユーザーにりてることができるは、このでモデルをします。

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

あなたのテーブルでは、

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

has_one

`has_one` アソシエーションは、のモデルと11のをしますが、セマンティクスは異なります。このけは、モデルのインスタスがのモデルの1つのインスタスをむか、またはしていることをします。

たとえば、アプリケーションのユーザーにアカウントが1つしかないは、のようにユーザーモデルをします。

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

アクティブレコードでは、`has_one` リレーションがある、アクティブレコードは、キーをつレコードが1つだけすることをします。

このでは、Accountテーブルでは、の`user_id`をつレコードが1つしかしません。じユーザーにしてもう1つのアカウントをけると、のエントリのキーが`null`になり、しいエントリがにされます。をするためにしいエントリのにしたでも、のエントリは`null`になります。

```
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]
```

くをっています

`has_many` アソシエーションは、のモデルとの1 `has_many` をします。このけは、に`belongs_to`けのにします。

このけは、モデルのインスタンスにのモデルのインスタンスがゼロあることをします。

たとえば、ユーザーとをむアプリケーションでは、ユーザーモデルはのようになります。

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

`Post`のテーブルは`belongs_to`とじです。に、`User`はスキーマのをとしません。

`User`されたすべてののリストをするは、をできますつまり、けオブジェクトにスコープをできます。

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

このタイプのけにより、`ActiveRecord`モデルはののモデルレコードにすることができます。な

```
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
```

このけがなければ、`Address`テーブルにこれらのキーがすべてありますが、このシナリオではアドレスは1つのエンティティ`Human`または`Company`にしかしていないため、いずれかのをつことになります。これは、のようになります。

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

`has_manythrough` アソシエーション

`has_many :through` アソシエーションは、のモデルと `many-to-many` をするためによくされます。この場合は、3のモデルをめることによって、モデルがのモデルの0のインスタンスとすることをす。

例えば、がにををするをえてみましょう。するはのようになります。

```
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_onethrough アソシエーション

`has_one :through association`は、のモデルと `one-to-one` をします。この場合は、3のモデルをめることによって、モデルがのモデルの1つのインスタンスとすることをす。

たとえば、`supplier`が1つの`account`をち、アカウントが1つのアカウントにけられている、サプライヤモデルはのようになります。

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

class Account < ApplicationRecord
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ApplicationRecord
  belongs_to :account
end
```

has_and_belongs_to_many アソシエーション

`has_and_belongs_to_many` アソシエーションは、モデルをさせずにのモデルと `many-to-many` をします。

たとえば、アプリケーションに `assemblies` と `parts` まれて `parts`、アセンブリにのパーツがあり、パーツがくのアセンブリにされているは、のようにモデルをできます。

```
class Assembly < ApplicationRecord
  has_and_belongs_to_many :parts
end
```

```
end

class Part < ApplicationRecord
  has_and_belongs_to_many :assemblies
end
```

は、モデルをそれにけるためにされます。もなは、とのとのをすることです。

。

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

では、モデルをけることができます。

```
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
```

のモデルはのようになります。

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

オンラインでアクティブレコードをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1820/アクティブレコード>

40: キャッシング

Examples

ロシアののキャッシング

キャッシュされたフラグメントをのキャッシュされたフラグメントにネストすることができます。これはRussian doll cachingとばれています。

Russian doll cachingのは、のがされた、のフラグメントをするとき、のすべてのフラグメントをできることです。

のセクションでしたように、キャッシュされたファイルがするレコードのupdated_atのがされた、キャッシュされたファイルはれになります。ただし、これによりフラグメントがネストされたキャッシュはれになりません。

たとえば、のようになります。

```
<% cache product do %>
  <%= render product.games %>
<% end %>
```

このビューをレンダリングします。

```
<% cache game do %>
  <%= render game %>
<% end %>
```

ゲームのがされた、 updated_atはのにされ、キャッシュがれになります。

ただし、オブジェクトのupdated_atはされなため、キャッシュのはれず、アプリはいデータをします。これをするために、々はモデルをタッチメソッドとびつける

```
class Product < ApplicationRecord
  has_many :games
end

class Game < ApplicationRecord
  belongs_to :product, touch: true
end
```

SQL キャッシュ

クエリキャッシュは、クエリによってされたセットをキャッシュするRailsです。 Railsがそのにしてクエリをした、データベースにしてクエリをするのではなく、キャッシュされたセットをします。

えば

```
class ProductsController < ApplicationController

  def index
    # Run a find query
    @products = Product.all

    ...

    # Run the same query again
    @products = Product.all
  end

end
```

クエリがデータベースにして2にされると、にはデータベースにヒットしません。がクエリからされたのときには、クエリキャッシュメモリにされ、2はメモリからされます。

ただし、クエリキャッシュはアクションのにされ、そのアクションのわりにされるため、アクションのみすることにするのがです。よりなでクエリをしたいは、レベルのキャッシュでうことができます。

フラグメントキャッシング

`Rails.cache`によってされている`Rails.cache`をして、リクエストにまたがってシリアライズなRubyオブジェクトをキャッシュすることができます。

キャッシュからのキーのをフェッチするには、`cache.read`し`cache.read`。

```
Rails.cache.read('city')
# => nil
```

`cache.write`をして、キャッシュにをきみます。

```
Rails.cache.write('city', 'Duckburgh')
Rails.cache.read('city')
# => 'Duckburgh'
```

わりに、`cache.fetch`をしてキャッシュからをみみ、オプションでがないはデフォルトをきむこともできます。

```
Rails.cache.fetch('user') do
  User.where(:is_awesome => true)
end
```

されたブロックのりは、されたキーのキャッシュにりてられてからされます。

キャッシュのをすることもできます。

```
Rails.cache.fetch('user', :expires_in => 30.minutes) do
  User.where(:is_awesome => true)
end
```

ページのキャッシュ

[ActionPack page_caching gem](#)をして々のページをキャッシュすることができます。これにより、1つののがHTMLファイルとしてされます。このファイルは、のにするのわりにされます。READMEにはながまれています。したら、コントローラーの`cache_page`クラス・メソッドをして、アクションのをキャッシュします。

```
class UsersController < ActionController::Base
  cache_page :index
end
```

`expire_page`をして、されたHTMLファイルをしてキャッシュのをします。

```
class UsersController < ActionController::Base
  cache_page :index

  def index
    @users = User.all
  end

  def create
    expire_page :action => :index
  end
end
```

`expire_page`のは、`url_for`および`friends`のをしています。

HTTPキャッシング

Rails> = 3には、HTTPキャッシュがしています。これは、`Cache-Control`および`ETag`ヘッダーをして、クライアントまたはCDNなどがページをキャッシュできるのをします。

コントローラーアクションでは、`expires_in`をして、そのアクションのキャッシングのをします。

```
def show
  @user = User.find params[:id]
  expires_in 30.minutes, :public => true
end
```

`expires_now`をして、しているクライアントまたはにキャッシュされたリソースのをします。

```
def show
  @users = User.find params[:id]
  expires_now if params[:id] == 1
end
```

アクションキャッシング

ページキャッシュとに、アクションキャッシュはページをキャッシュします。いは、キャッシュがされるにフィルタがされるに、リクエストがRailsスタックにヒットすることです。それはRailsから[actionpack-action_caching gem](#)にされました。

なは、がなアクションのキャッシュです。

```
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
```

オプションには、`:expires_in`、`custom :cache_path` のを々にキャッシュするのあるアクション、および/`:unless`アクションをキャッシュするタイミングをし:`unless` `:if` / `:unless`

```
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
```

レイアウトにコンテンツがあるは、`layout: false`をしてアクションコンテンツのみをキャッシュし`layout: false`。

オンラインでキャッシングをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2833/キャッシング>

41: クラス

これはなことのようですが、クラスでがまると、をかけてしていただきしています。

Examples

モデルクラス

```
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
```

モデルは、

- をする
- データの
- スcopeとメソッドをしてデータへのアクセスをする
- データのにするをします。

レベルでは、モデルはドメインをし、をします。

サービスクラス

コントローラは、アプリケーションのエントリーポイントです。しかし、なエントリーポイントで

はありません。このロジックをからアクセスできるようにしたいとえています

- レイクタスク
- バックグラウンドジョブ
- コンソール
- テスト

コントローラーにロジックをげれば、これらのからアクセスすることはできません。そこで、「スキニーコントローラ、ファットモデル」アプローチをして、そのロジックをモデルにしてみましよう。しかし、どちらのロジックに `User`、`Cart`、`Product` モデルがまれている、どこにそれをくがありますか

`ActiveRecord::Base` からしたクラスは、すでにくのをっています。これは、クエリインターフェイス、くをします。モデルにさらにくのコードをすると、ものパブリックメソッドがすぐにされなくなります。

サービスはのRubyオブジェクトです。そのクラスはのクラスからするはありません。そのは、たとえば、である `CreateUserAccount` のではなく `UserCreation` または `UserCreationService`。これは `app/services` ディレクトリにあります。このディレクトリはですありますが、Railsはでクラスをみみします。

サービスオブジェクトは1つのことをいます

サービスオブジェクトメソッドオブジェクトは、1つのアクションをします。そのアクションをするビジネスロジックをします。にをします。

```
# app/services/accept_invite.rb
class AcceptInvite
  def self.call(invite, user)
    invite.accept!(user)
    UserMailer.invite_accepted(invite).deliver
  end
end
```

がう3つのはのとおりです。

サービスは、`app/services` directory ます。はビジネスロジックいドメインのためにサブディレクトリをすることをめします。えは

- `app/services/invite/accept.rb` ファイルは `Invite::Accept` をし、`app/services/invite/create.rb` は `Invite::Create` をし `Invite::Create`
- サービスはでまりますサービスでわらない `ApproveTransaction`、`SendTestNewsletter`、`ImportUsersFromCsv`
- サービスは `call` メソッドにし `call`。のをするとしになります `ApproveTransaction.approve()` はうまくみません。また、`call` メソッドは、`lambda`、`procs`、および `method` オブジェクトのメソッドです。

サービスオブジェクトはのアプリケーションがをするかをします

サービスディレクトリをれば、のアプリケーションがをするかをすることができます

ApproveTransaction、 CancelTransaction、 BlockAccount、 SendTransactionApprovalReminder ...

サービスオブジェクトをすばやくて、ビジネスロジックがどのようなものかを知っています。はコントローラ、 ActiveRecordモデルのコールバック、オブザーバをして"トランザクションの"にすることをすることはありません。

クリーンアップモデルとコントローラ

コントローラはリクエストparams、 session、 cookiesをにし、サービスにし、サービスによってリダイレクトまたはレンダリングします。

```
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
```

モデルはアソシエーション、スコープ、のみをいいます。

```
class Invite < ActiveRecord::Base
  def accept!(user, time=Time.now)
    update_attributes!(
      accepted_by_user_id: user.id,
      accepted_at: time
    )
  end
end
```

これにより、モデルとコントローラのテストとがにになりました。

サービスクラスをする

アクションがの1つのをたす、サービスオブジェクトにします。

- はであるえば、のわりにをじる
- このアクションは、のモデルにまたがっていますたとえば、 Order、 CreditCard、 および Customerオブジェクトをしたの
- このアクションは、サービスソーシャルネットワークへのなど
- このアクションは、となるモデルのなではありませんえば、にいデータをするなど。
- アクションをするにはのがありますたとえば、アクセストークンまたはパスワードによる。

ソース

[Adam Niedzielski Blog](#)

Brew House Blog

コードのブログ

オンラインでクラスをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7623/クラス>

42: ターボリンク

き

Turbolinksは、WebアプリケーションをよりくナビゲートするためのJavaScriptライブラリです。リンクをたどると、Turbolinksはにページをし、<body>でスワップし、<head>をマージします。

ルールは、にターボリンクをにえることができます。しかし、これは、つけにくいバグのとなるがあるため、よくっておくべきなです。

なりみ

- `turbolinks:load` `document.ready` イベントのわりに `turbolinks:load` イベントにバインドする
- `data-turbolinks-false` をして、リンクごとにターボリンクをにします。
- `data-turbolinks-permanent` をして、ページみみのをし、キャッシュのバグをします。

ターボリンクのについては、 [githubリポジトリ](#) をご覧ください。

このドキュメンテーションのくは、githubリポジトリの `turbolink` のドキュメントをしてくれたたちのによるものです。

Examples

ターボリンクのページロードのへのバインディング

ターボリンクでは、をするのアプローチ

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

しません。ターボリンクをしている、 `$(document).ready()` イベントはのページのみみに1だけします。そのから、ユーザーがあなたのウェブサイトのリンクをクリックするたびに、ターボリンクがリンククリックイベントをし、<body>タグをきえて<head>タグをマージするようにajaxリクエストをいます。プロセスが、ターボリンクのの「」のをきこします。したがって、のの `document.ready()` をするわりに、 `turbolink` のイベントにバインドするがあります。

```
// pure js  
document.addEventListener("turbolinks:load", function() {  
  // awesome code  
});  
  
// jQuery  
$(document).on('turbolinks:load', function() {
```

```
// your code
});
```

のリンクでターボリンクをにする

のリンクでターボリンクをにすることはにです。 [のturbolinksの](#)によると

ターボリンクは、linkまたはそののいずれかにdata-turbolinks = "false"とをけることによつて、リンクごとににすることが出来ます。

```
// 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>
```

アプリケーションの

アプリケーションのは、ターボリンクのリンクをクリックするか、

```
Turbolinks.visit(location)
```

デフォルトでは、は 'advance' アクションをします。もっとわかりやすいことに、のデフォルトのは、「」パラメータによってされるページにむことです。ページがされるたびに、turboLinkは history.pushState をしてブラウザの新しいエントリをプッシュします。ターボリンクはをしてなりキャッシュからページをみもうとするため、はです。これにより、にれるページのページレンダリングがにになります。

ただし、ヒストリをスタックにプッシュしないでをれるは、のようにvisitで 'replace' アクションを出来ます。

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

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

これにより、ヒストリスタックのがしいページにきえられ、スタックのアイテムのはされません。

また、ユーザーがブラウザの「む」ボタンまたは「る」ボタンをクリックしたとしてする [リスト](#) をする「」アクションもあります。Turbolinksでは、これらのタイプのイベントをにし、ユーザーがでデフォルトのをざんすることをしています。

にをキャンセルする

Turbolinksは、のをめるためにできるイベントリスナーをします。 `turbolinks:before-visit` イベントをいて、 `turbolinks:before-visit` にされます。

イベントハンドラでは、のものをできます。

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

または

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

のをする。は、のでりすことができます。

```
event.preventDefault()
```

[のturbolinks docs](#)によると

はりすことができず、ターボリンクをしなない。

ページのみみのの

のようなをえてみましょう。ユーザーがソーシャルメディアのWebサイトのであり、ユーザーがのユーザーとだちになり、ターボリンクをしてページのみみをするとします。サイトのページのには、ユーザーがっているのをすがあります。あなたのサイトをしていると3のがいるとします。しいがされるたびに、カウンターをするjavascriptがいくつかあります。あなたがちょうどしいをしたことをして、あなたのjavascriptがにし、ページののにのがされたことをしてみましょう。4.に、ブラウザのるボタンをクリックしたとします。ページがみまれると、が4いるのにカウンターに3とされます。

これはなであり、ターボリンクがをしているです。このがするは、ユーザーがるボタンをクリックするとターボリンクがにキャッシュからページをみむためです。キャッシュされたページはにデータベースでされるとはりません。

このをするには、カウントをidの "friend-count"の<div>タグのにレンダリングするとします。

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

`data-turbolinks-permanent` をすることで、ターボリンクにのをページみみにするようにします。の
はう

なをするには、HTML IDをえ、データターボリンクをにをけます。レンダリングのに
、Turbolinksはなすべてのをidでマッチングし、のページからしいページにそれらをし
、データとイベントリスナーをします。

オンラインでターボリンクをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9331/ターボリンク>

43: デコレータパターン

デコレータパターンをすると、ベースオブジェクトにをえずに、にじてオブジェクトのをまたはできます。

これは、`stdlib`をするなRubyやDraperのようななをしてできます。

Examples

SimpleDelegatorをしたモデルの

ほとんどのRailsは、テンプレートのモデルをすることからめます。

```
<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>
```

くのデータをつモデルの、これはすぐになになり、あるテンプレートからのテンプレートにロジックをコピー・ペーストすることにつながります。

このでは、`stdlib`のSimpleDelegatorをしています。

SimpleDelegatorオブジェクトへのすべてののは、デフォルトでオブジェクトにされます。プレゼンテーションロジックでのメソッドをオーバーライドすることも、このビューにのしいメソッドをすることもできます。

SimpleDelegator 2つのメソッドをします。 `__setobj__`にされているものを、およびする `__getobj__` そのオブジェクトをします。

```
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
```

いくつかのデコレータは、このをワイヤリングするためににしていますが、ページのオブジェクトをすることによってプレゼンテーションロジックがどこからているのかをよりにすることができます。

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

ビュー・オブジェクトへのをデコレータにすことで、プレゼンテーション・ロジックをみむことなく、ビュー・ヘルパのりのにアクセスすることができます。

これで、ビューテンプレートはページにデータをすることのみにがあり、はるかにです。

Draperをったモデルの

Draperは、コンベンションによってモデルをデコレータとにマッチングさせます。

```
# 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
```

ActiveRecordオブジェクトをむ@userをすると、@userで#decorateをびすことによってデコレータにアクセスしたり、のものにしたいにDraperクラスをしてデコレータにアクセスできます。

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

オンラインでデコレータパターンをむ <https://riptutorial.com/ja/ruby-on-rails/topic/5694/デコレータパターン>

44: デバッグ

Examples

Railsアプリケーションのデバッグ

アプリケーションのロジックとデータのねをするには、アプリケーションをデバッグできることがにです。なバグをするのにち、プログラミングのとコードののにをもちます。2つのデバッグのとして、[デバッガ](#) `ruby 1.9.2`と`1.9.3`と[byebug](#) `ruby> = 2.x`があります。

`.rb`ファイルをデバッグするには、のをします。

1. `debugger`または`byebug`を`byebug`の`development`グループにGemfile
2. `bundle install``bundle install`
3. `debugger`または`byebug`をブレークポイントとしてする
4. コードをするか、リクエストする
5. されたブレークポイントでしたルール・サーバー・ログをしてください
6. こので、あなたはサーバを`rails console`ようにい、とパラメータのをチェックすることができます
7. のにするには、`↑` `next`と`enter next`をし`enter`
8. タイプ`c`をして`enter c`をす

`.html.erb`ファイルをデバッグ`.html.erb`、ブレークポイントは`<% debugger %>`としてされ`<% debugger %>`

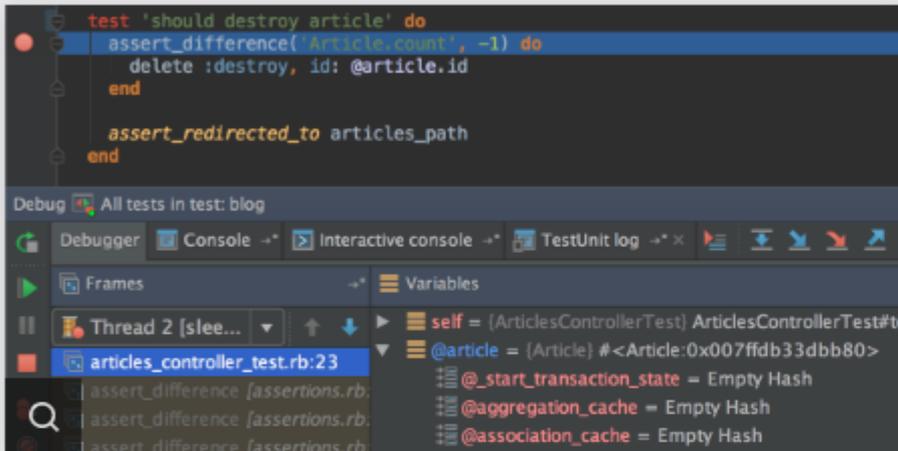
IDEでのデバッグ

れたIDEはすべて、ブレークポイント、、にするRubyおよびRailsアプリケーションをインタラクティブにデバッグするためのGUIをしています。

たとえば、Ruby IDEののRubyMineのデバッグの1つ

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 view.
- 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.

Ruby on Rails をくデバッグする+のアドバイス

をけることにより、デバッグはをじてめよりもはるかにである`print` ログステートメント、およびほとんどのバグのために、そのにはるかにのようなIRBデバッガくよりも`pry`または`byebug`。これらのツールはあなたののステップであってはなりません。

Ruby / Rails をすばやくデバッグする

1. メソッド `Exception` を `.inspect` せ、そのを `.inspect` する

RubyにRailsコードをデバッグするもいは、メソッドやオブジェクト `foo` で `.inspect` をびしながら、コードのパスによってをさ `raise` です。

```
raise foo.inspect
```

のコードで `raise` は、コードのをする `Exception` をトリガし、デバッグしようとしているのオブジェクト/メソッドつまり `foo` にする `.inspect` をにむエラーメッセージをします。

このテクニックは、オブジェクトやメソッドをくべるのにです えば、`nil`はあり`nil`んか、また、のコンテキストでコードのがまったくされているかどうかをすぐにするのにです。

2. フォールバックようルビー—**IRB**デバッガをし `byebug` または `pry`

あなたのコードのフローのについてのをっていたにのみ、あなたのようなルビーのIRBデバッガへのをしてください `pry` や `byebug` あなたのパスのオブジェクトのにさらにくりげることができます。

`byebug` `gem`をRailsでのデバッグにするには

1. `gem 'byebug'`ご **Gemfile**にグループ
2. `bundle install``bundle install`
3. するには、べたいコードのパスのに `byebug` というフレーズをします。

この`byebug`をすると、コードのルビーIRBセッションがき、コードのにそののオブジェクトのにアクセスできるようになります。

ByebugのようなIRBデバッガは、されるコードのをくするのにです。しかし、エラーをさせるのにべてきはがかけりますので、ほとんどのではのステップではありません。

のアドバイス

をデバッグしようとしているときは、にアドバイスをしてください **@\$ing** エラーメッセージ

RTFM

つまり、のにエラーメッセージをかつにみて、をえようとしているのかをすることをします。デバッグするときは、エラーメッセージをむときに、のなをこのでねます。

1. どのクラスがエラーをしていますか つまり、はしいオブジェクトクラスをっているのかまたはオブジェクトが`nil`ですか
2. エラーはどのようなでされますか つまり、メソッドの;この/オブジェクトのクラスでこのメソッドをびせますか
3. に、の2つのからできるものを持って、どのようなコードをべるべきですか えておいてください スタックトレースののコードは、ずしもがするではありません。

スタックトレースでは、あなたのプロジェクトからるコードののにをうべき `app/...` えば、**Rails** をっているなら、 `app/...`まる。のコードでがしているの99。

このですることがなをするために ...

たとえば、くのをさせる**Ruby**エラーメッセージ

あるでそのようなコードをします

```
@foo = Foo.new  
  
...  
  
@foo.bar
```

のようなエラーがされます。

undefined method "bar" for Nil:nilClass

はこのエラーをて、メソッド`bar`がされていないというがあるとえます。 そうではありません。このエラーでは、なはのとおりです。

```
for Nil:nilClass
```

for Nil:nilClassは`@foo`が**Nil**であることをします。 `@foo`は`Foo`インスタンスではありませんあなたは`Nil`オブジェクトをっています。このエラーがされたときは、にルビがクラス`Nil`オブジェクトにしてメソッド`bar`がしないことをえようとしています。 たちがクラス`Foo`オブジェクトにして`Nil`はないメソッドをしようとしているからです。

ながら、このエラーがどのようにされているのか `undefined method "bar" for Nil:nilClass`、このエラーは、`bar`が`undefined`とえるがあり`bar`。 くまないと、このエラーはがって`Foo`の`bar`メソッドのをりげ、オブジェクトがったクラスこのは`nil`であることをするエラーのをにいてしまいます。エラーメッセージをむことのでにできるいです。

デバッグをするに、ずエラーメッセージをくおみください。それはあなたは、エラーがするかもしれないとうのコードのいずれかのスタックトレースまたはラインにsleuthingするに、ず、そのメソッドその、のエラーメッセージでオブジェクトのクラスをしてください。それらの5はあなたに5のをうことができます。

tl; dr ログをべないでください。わりにをさせてください。デバッグするにエラーをくんで、ウサギのをけてください。

pry をって ruby-on-rails アプリケーションをデバッグする

pry は、Ruby アプリケーションをデバッグするためにできるなツールです。このをって Ruby-on-Rails アプリケーションをするのは、とてもです。

セットアップ

pry でアプリケーションのデバッグをするには

- アプリケーションの Gemfile に `gem 'pry'` Gemfile `gem 'pry'` をしてバンドルします

```
group :development, :test do
  gem 'pry'
end
```

- ターミナルコンソールでアプリケーションのルートディレクトリにし、`bundle install` ます。アプリケーションのどこにでもいめることができます。

つかいます

アプリケーションで pry をすると、デバッグにしたいブレイクポイントに `binding.pry` をめるだけです。 `binding.pry` ブレイクポイントは、Ruby インタープリタ `app/controllers`、`app/models`、`app/views` ファイルによってされるアプリケーションのどこにでも `binding.pry` ます。

i コントローラのデバッグ

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def show
    use_id = params[:id]
    // breakpoint to inspect if the action is receiving param as expected
    binding.pry
    @user = User.find(user_id)
    respond_to do |format|
      format.html
    end
  end
end
```

このでは、ページルーティングにアクセスして `UsersController` アクションを `show` しようとする、レールサーバーがブレイクポイントでコンソールをしてします。 `params` オブジェクトをべて、そ

のブレークポイントから`User`モデルのActiveRecordクエリをできます

iiビューのデバッグ

`app / views / users / show.html.haml`

```
%table
  %tbody
    %tr
      %td ID
      %td= @user.id
    %tr
      %td email
      %td= @user.email
    %tr
      %td logged in ?
      %td
        - binding.pry
        - if @user.logged_in?
          %p= "Logged in"
        - else
          %p= "Logged out"
```

このでは、`users/show`ページがクライアントのブラウザにされるに、railsサーバであらかじめコンパイルされているとき、`pry`コンソールでブレークポイントがします。このブレークポイントは、`@user.logged_in?`しさをデバッグできます `@user.logged_in?`それがっているとき。

iiモデルのデバッグ

```
app/models/user.rb

class User < ActiveRecord::Base
  def full_name
    binding.pry
    "#{self.first_name} #{self.last_name}"
  end
end
```

このでは、このメソッドがアプリケーションのどこからでもびされたときに、ブレークポイントをして`User`モデルのインスタンスメソッド`full_name`をデバッグすることができます。

として、`pry`はなとなデバッグガイドラインをえたルールアプリケーションのなデバッグツールです。これをしてみてください。

オンラインでデバッグをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3877/デバッグ>

45: デフォルトの Rails アプリケーションをする

き

ここではをどのようにするのかをします。そうすれば、かがするのではなく、らがむできるように rails s タイプすることができます。

Examples

ローカルマシンでする

、することによってレールがされるとき。これは、 development デフォルトをするだけです

```
rails s
```

のは、 -e をしてできます。

```
rails s -e test
```

どちらがテストをするか。

デフォルトのは ~/.bashrc ファイルをし、のをすることででできます

```
export RAILS_ENV=production in your
```

サーバーでする

Passenger をしているリモートサーバーでしているは、するに apache.conf をします。たとえば、このケースでは RailsEnv production がされます。

```
<VirtualHost *:80>
  ServerName application_name.rails.local
  DocumentRoot "/Users/rails/application_name/public"
  RailsEnv production ## This is the default
</VirtualHost>
```

オンラインでデフォルトの Rails アプリケーションをするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9915/デフォルトのrailsアプリケーションをする>

46: デフォルトのタイムゾーンをする

`config.active_record.default_timezone`は、データベースからとをするときに、`Time.locallocal`にされているまたは`Time.utcutc`にされているをするかどうかをします。デフォルトは`utc`です。 <http://guides.rubyonrails.org/configuring.html>

Railsのタイムゾーンをしたいが、**Active Record**をデータベースに**UTC**でしけたいは、

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

あなたは**Rails**のタイムゾーンをし、このタイムゾーンでアクティブレコードストアのをって、したい

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

データベースにUTCのでをするに、には2、さらには3とえるべきです。

`application.rb`をした、**Rails**サーバーをするのをれないでください。

`config.active_record.default_timezone`は2つのしか`config.active_record.default_timezone`できないことに`config.active_record.default_timezone`ください

- **local** `config.time_zone`されたタイムゾーンに
- **utc** UTCに

なすべてのタイムゾーンをつけるはのとおりです

```
rake time:zones:all
```

Examples

Railsのタイムゾーンをしますが、**Active Record**を**UTC**でデータベースにしけます

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
```

Railsのタイムゾーンをし、このタイムゾーンに**Active Record**のをする

```
# application.rb
config.time_zone = 'Eastern Time (US & Canada)'
config.active_record.default_timezone = :local
```

オンラインでデフォルトのタイムゾーンをするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3367/デフォルトのタイムゾーンをする>

47: ドッカーのルール

き

このチュートリアルでは、Dockerがインストールされ、Railsアプリ

Examples

ドッカーとドッカーの

まず、Dockerfileをするがあります。いはNick Janetakisのこの[ブログ](#)でつけることができます。

このコードには、にドッカーマシンでされるスクリプトがまれています。このため、なライブラリをすべてインストールし、PumaRoR dev server

```
# Use the barebones version of Ruby 2.3.
FROM ruby:2.3.0-slim

# Optionally set a maintainer name to let people know who made this image.
MAINTAINER Nick Janetakis <nick.janetakis@gmail.com>

# Install dependencies:
# - build-essential: To ensure certain gems can be compiled
# - nodejs: Compile assets
# - libpq-dev: Communicate with postgres through the postgres gem
RUN apt-get update && apt-get install -qq -y --no-install-recommends \
    build-essential nodejs libpq-dev git

# Set an environment variable to store where the app is installed to inside
# of the Docker image. The name matches the project name out of convention only.
ENV INSTALL_PATH /mh-backend
RUN mkdir -p $INSTALL_PATH

# This sets the context of where commands will be running in and is documented
# on Docker's website extensively.
WORKDIR $INSTALL_PATH

# We want binstubs to be available so we can directly call sidekiq and
# potentially other binaries as command overrides without depending on
# bundle exec.
COPY Gemfile* $INSTALL_PATH/

ENV BUNDLE_GEMFILE $INSTALL_PATH/Gemfile
ENV BUNDLE_JOBS 2
ENV BUNDLE_PATH /gembox

RUN bundle install

# Copy in the application code from your work station at the current directory
# over to the working directory.
COPY . .
```

```
# Ensure the static assets are exposed to a volume so that nginx can read
# in these values later.
VOLUME ["$INSTALL_PATH/public"]

ENV RAILS_LOG_TO_STDOUT true

# The default command that gets run will be to start the Puma server.
CMD bundle exec puma -C config/puma.rb
```

また、**docker-compose**をします。そのために、`docker-compose.yml`をします。このファイルのは、**Rails**とのよりもドッカーのチュートリアルになりますので、ここではしません。

```
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
```

これらの2つのファイルだけで、`docker-compose up`をするだけです。 `docker-compose up` できます

オンラインでドッカーのルールをむ <https://riptutorial.com/ja/ruby-on-rails/topic/10933>/ドッカーのルール

48: ハイパーループをったReact.jsとRailsの

き

このトピックでは、 [Hyperloop gem](#)をしたReact.jsとRailsのについてします。

ここでわれていないのアプローチは、レールまたはreact_on_railsのです。

コンポーネントクラスは、にのjavascriptコンポーネントクラスをするだけです。

Rubyのコンポーネントクラスからjavascriptのコンポーネントやライブラリにアクセスすることもできます。

Hyperloopはビューサーバーを「プリレンダー」して、ビューはERBまたはHAMLテンプレートのようにロードされます。クライアントにロードされると、ユーザーからの、HTTPリクエスト、またはWebソケットデータのために、のとしてDOMが々にされ、DOMが々にされます。

コンポーネントのほかに、Hyperloopにはをするストア、ビジネスロジックをカプセルする、ARをしてクライアントのActiveRecordモデルにアクセスできるモデルがあります。

はこちら <http://ruby-hyperloop.io/>

Examples

Rails アプリケーションになコンポーネントルビーでかてているをする

1. あなたのレールにハイパーループのをする4.0 - 5.1Gemfile
2. `bundle install`
3. ハイパーループマニフェストをapplication.jsファイルにします。

```
// app/assets/javascripts/application.js
...
//= hyperloop-loader
```

4. コンポーネントをし、 `hyperloop/components`ディレクトリにします

```
# 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. コンポーネントはビューのようにします。コントローラの`render_component`メソッドをして

「マウント」されます。

```
# somewhere in a controller:
...
def hello_world
  render_component # renders HelloWorld based on method name
end
```

コンポーネントパラメータの

```
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 タグ

```
# HTML tags are built in and are UPCASE
class HTMLExample < Hyperloop::Component
  render do
    DIV do
      SPAN { "Hello There" }
      SPAN { "Welcome to the Machine!" }
    end
  end
end
```

イベントハンドラ

```
# 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
```

```
# 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
end
```

[Hyperloop :: Stores](#)をしてコンポーネントでできることにしてください

コールバック

```
# 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
```

オンラインでハイパーloopを使ったReact.jsとRailsのをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9809/ハイパーloopを使ったreact-jsとrailsの>

49: ビュー

Examples

テンプレートは、レンダリングプロセスをよりしやすいにします。パーシャルアールをすると、テンプレートからコードをしてファイルをし、テンプレートですることができます。

をするには、`_form.html.erb`というアンダースコアでまるしいファイルをしします。

ビューのとしてをレンダリングするには、ビューの`render`メソッドをしします。 `<%= render "form" %>`

- レンダリングにアンダースコアはされていることにしてください
- パーツは、のフォルダにあるはそのパスをしてレンダリングするがあります

をにローカルとしてすには、のをしします。

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

コードは、まったくじコード **DRY** をするがあるにもです。

えば、するために`<head>`、コードをにの`_html_header.html.erb`、あなたのし`<head>`コードをするために、とがとするたびをレンダリング `<%= render 'html_header' %>`。

オブジェクトパーシャル

`to_partial_path`するオブジェクトは、`<%= render @post %>`こともできます。デフォルトでは、ActiveRecordモデルのは`posts/post`ようになりますので、に`@post`レンダリング`@post`と、ファイルの`views/posts/_post.html.erb`がレンダリングされます。

ローカルのき`post`がにりてられます。、`<%= render @post %>`は`<%= render 'posts/post', post: @post %>`いです。

`<%= render @posts %>`ように、`to_partial_path`するオブジェクトのコレクションをすることもでき`<%= render @posts %>`。アイテムはしてレンダリングされます。

グローバルパーシャル

なパスをせずにもどこでもできるグローバルパーシャルをするには、パーシャルを`views/application`パスにする`views/application`ます。のは、このをするためににされています。

たとえば、これはグローバルな`app/views/application/_html_header.html.erb`:

このなグローバルなをレンダリングするには、`<%= render 'html_header' %>`

AssetTagHelper

ほとんどの、ヘルパーをみみたいは、ルールにに `css / js / images` をかつにリンクさせるためです。

イメージヘルパー

image_path

これにより、`app/assets/images` アセットへのパスがされます。

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

image_url

これにより、`app/assets/images` アセットへのなURLがされます。

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

image_tag

`` タグをソースセットにめるには、このヘルパーをします。

```
image_tag("icon.png") # => 
```

JavaScriptヘルパー

javascript_include_tag

ビューにJavaScriptファイルをめる。

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

javascript_path

JavaScriptファイルのパスがされます。

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

javascript_url

これはJavaScriptファイルのなURLをします。

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

スタイルシートヘルパー

stylesheet_link_tag

ビューにCSSファイルをめる。

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

stylesheet_path

これにより、スタイルシートアセットのパスがされます。

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

stylesheet_url

あなたのスタイルシートアセットのなURLをします。

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

しいルールアプリをするときには、に `app/views/layouts/application.html.erb` 2つのヘルパーがあり `app/views/layouts/application.html.erb`

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

これは、

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

RailsはMVCパターンによってViewsされます。Viewsは、あなたの"テンプレート"があなたのアクションのためにあるです。

たとえば、コントローラ`articles_controller.rb`があるとします。このコントローラの、`app/views/articles`というビューのフォルダがあります。

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

このにより、コントローラのフォルダをつことができます。コントローラでアクションをびすと、なビューがにレンダリングされます。

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

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

ビューでのHTMLコードのきえ

ランタイムにページにされるhtmlコンテンツをしたいとっていた、そのためのにいがルールにあります。 **content_for**とばれるものがあります。これによって、ブロックをルールビューにすことができます。のをしてください。

コンテンツをする

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

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

HAML - あなたのですの

HAMLHTMLマークアップは、ビューのHTMLをしてデザインするためのしくエレガントなです。タグをいたりじたりするわりに、HAMLはページのにインデントをいます。に、のにするがあるは、タブストップを1つしてインデントします。HAMLではタブとがなので、にじのタブをするようにしてください。

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

そして、ハムで

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

レイアウトのは、HTMLとERBをするよりはるかにです。

インストール

にをインストールする

```
gem install haml
```

gemfileにをする

```
gem "haml"
```

HTML / ERBのわりにHAMLをするは、ビューのファイルを something.html.haml から something.html.erb にきえてください。

クイックティップス

divのようななはにくことができます

HTML

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

ハムル

```
%div.myclass
```

HAML、

```
.myclass
```

HTML

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

ハムル

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

ルビコードの

=と - をけてルビコードをすることができます。

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

=でまるコードがされ、にめまれます。

-でまるコードはされますが、ドキュメントにはされません。

なドキュメント

HAMLはにですが、になので、[ドキュメントをむ](#)ことをおめします。

[オンラインでビューをむ](https://riptutorial.com/ja/ruby-on-rails/topic/850/ビュー) <https://riptutorial.com/ja/ruby-on-rails/topic/850/ビュー>

50: ファイルのアップロード

Examples

Carrierwaveをしたのファイルアップロード

ファイルアップロードをRailsですることはです。まず、アップロードをするためのプラグインがです。もなものは**Carrierwave**と**Paperclip**です。どちらもはていて、ドキュメントはです

Carrierwaveをつたなアバターアップロードのをてみましょう

Carrierwaveをbundle install、コンソールにしてください

```
$ rails generate uploader ProfileUploader
```

これにより/app/uploaders/profile_uploader.rbにあるファイルがされます

ここでは、ストレージつまりローカルまたはクラウドをしたり、のをしたりつまり、MiniMagickをしてサムをする、サーバーのホワイトリストをすることができます

に、user_picのtypeでしいをし、user.rbモデルでuploaderをマウントします。

```
mount_uploader :user_pic, ProfileUploader
```

に、アバターをアップロードするためのフォームをしますユーザーのビューかもしれません

```
<% form_for @user, html: { multipart: true } do |f| %>
  <%= f.file_field :user_pic, accept: 'image/png, image/jpeg' %>
  <%= f.submit "update profile pic", class: "btn" %>
<% end %>
```

フォームがアップロードをできるように{multiparttrue}をめてください。Acceptは、クライアントのホワイトリストをするオプションです。

アバターをするには、に

```
<%= image_tag @user.user_pic.url %>
```

ネストされたモデル . のアップロード

のアップロードをするは、まずしいモデルをしてをする

たとえば、Productモデルでのがなをえてみましょう。しいモデルをし、それをあなたのモデルにbelongs_toせる

```
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`は、ネストされたフォームをできるので、しいファイルをアップロードし、をし、をのフォームからできるため、です

に、ビュー/でフォームをし、

```
<%= 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 %>
```

コントローラーはもなものではありません。しいコントローラーをしたくないは、コントローラーにしいコントローラーをしてください

```
# 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
```

すべてのをビューにする

```
<% @product.product_photos.each do |i| %>  
  <%= image_tag i.image.url, class: 'img-rounded' %>  
<% end %>
```

オンラインでファイルのアップロードをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2831/ファイルのアップロード>

51: フォームヘルパー

き

Railsは、フォームマークアップをするためのビューヘルパーをします。

- `date`、`datetime`、`datetime-local`、`time`、`month`、`week`などのタイプは、Firefoxではしません。
- `input<type="telephone">`はSafari 8でのみします。
- `input<type="email">`がSafariでしません

Examples

フォームをする

`form_tag`ヘルパーをしてフォームをできます

```
<%= form_tag do %>
  Form contents
<% end %>
```

これにより、のHTMLがされます。

```
<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>
```

このフォームタグは、`hidden`フィールドをしました。これは、フォームがフォームなしでにできないためです。

`authenticity_token`というの2のフィールドは、`cross-site request forgery`にしてをします。

フォームの

フォームをするには、のコードをします

```
<%= form_tag("/search", method: "get") do %>
  <%= label_tag(:q, "Search for:") %>
  <%= text_field_tag(:q) %>
  <%= submit_tag("Search") %>
<% end %>
```

- `form_tag` フォームをするためのデフォルトのヘルパーです。のパラメータ、`/search`はアク

ション、2のパラメータはHTTPメソッドをします。フォームの、に`get`メソッドをすることが

- `label_tag` このヘルパーは`html <label>`タグをします。
- `text_field_tag` これは、`text`タイプのをし`text`
- `submit_tag` これは、`submit`のをします。

フォームのヘルパー

チェックボックス

```
<%= 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") %>
```

これはのHTMLをします

```
<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>
```

ラジオボタン

```
<%= 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") %>
```

これにより、のHTMLがされます。

```
<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>
```

テキスト

きなテキストボックスをするには、`text_area_tag`をすることをめします

```
<%= text_area_tag(:message, "This is a longer text field", size: "25x6") %>
```

これにより、のHTMLがされます

```
<textarea id="message" name="message" cols="25" rows="6">This is a longer text
field</textarea>
```

フィールド

これは `input<type="number">` をし `input<type="number">`

```
<%= number_field :product, :rating %>
```

のをするには、 `in:` オプションをできます

```
<%= number_field :product, :rating, in: 1..10 %>
```

パスワードフィールド

によっては、ユーザーがしたをマスクするがあります。これにより、 `<input type="password">`

```
<%= password_field_tag (:password) %>
```

メールフィールド

これにより、 `<input type="email">`

```
<%= email_field (:user, :email) %>
```

フィールド

これにより、 `<input type="tel">` がされます。

```
<%= telephone_field :user, :phone %>
```

ヘルパー

- `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) %>
```

ちる

な @models = Model.all select_tag "models"、 options_from_collection_for_select @models、 "id"、 "name"、 {}

これはのHTMLをしますDavid

のはoptionsで、 {multiplefalse、 disabledfalse、 include_blankfalse、 promptfalse}をけけます。

よりくのがわかります http://apidock.com/rails/ActionView/Helpers/FormTagHelper/select_tag

オンラインでフォームヘルパーをむ <https://riptutorial.com/ja/ruby-on-rails/topic/4509/フォームヘルパー>

52: フレンドリーなID

き

FriendlyIdはActive Recordのスラギングプラグマとパーマリンクプラグインの「Swiss Army Bulldozer」です。これは、あなたがきれいなURLをし、にやさしいをのよううことができます。 FriendlyIdをすると、アプリケーションでのようなURLをにできるようになります。

<http://example.com/states/washington>

Examples

Rails クイックスタート

```
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
```

app / models / user.rb をする

```
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')
```

シーケンスをするのではなく、レコードをにするためにできるのslugのリストをにできるしい「」。えは

```
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
```

Friendly_id gemをしてslugのさをしますか

```
def normalize_friendly_id(string)
  super[0..40]
end
```

オンラインでフレンドリーなIDをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9664/フレンドリーなid>

53: モデル AASM

Examples

AASMによる

は、をむモデルをすることになり、そのはオブジェクトのにします。

AASMは、オブジェクトのプロセスをにすることにして、なステートマシンイネーブラーライブラリです。

あなたのモデルにこのようなことをたせることは、Railsのベストプラクティスの1つであるFat Model、Skinny Controllerのアイデアとびついています。モデルは、その、およびそれらによってきこされるイベントのをするがあるのモデルです。

Gemfileにインストールするには

```
gem 'aasm'
```

ユーザーがのためにをするAppをえてみましょう。

```
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
```

```
private

def set_payment_date
  update payed_at: Time.zone.now
end

end
```

ただし、Quoteクラスのは、プロセスにです。

あなたは、ののようにのもの、のもの、あるいはのものといえることができます。たとえば、い、などです。のはあなたです。なからると、のは、あなたのがずのであり、イベントとのリンクがになるため、よりです。

あなたはどのをするかしてください。あなたはRubyやRuby on Railsのみキーワード `valid`、`end`、`being`などをしないことをする`being`あります。

とをしたら、AASMによってされたいいくつかのメソッドにアクセスできるようになりました。

えば

```
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.
```

イベントにがあるのがわかるように、このは、イベントびしにがどのようにするかをします。のためイベントがな、エラーがします。

イベントとトランジションには、にもいくつかのコールバックがあります。

```
guard: product_delivered?
```

`product_delivered?`びし`product_delivered?`メソッドはブールをします。それがになると、はされず、のができない、はしない。

```
success: :reverse_charges
```

そのがにわれた、`:reverse_charges`メソッドがびされます。

AASMには、このプロセスでコールバックがえているいくつかのがありますが、これはでのモデルをするのにちます。

オンラインでモデルAASMをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7826/モデル-aasm>

54: モンゴイド

Examples

インストール

まず、あなたのGemfile Mongoidをしてください

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

bundle install ます。またはにする

```
$ gem install mongoid
```

インストール、ジェネレータをしてファイルをしませ

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

ファイル (myapp)/config/mongoid.yml をしませ。

モデルの

をしてモデルをしませ User とぶことができます。

```
$ rails g model User
```

app/models/user.rb というファイルがされませ

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

これはあなたがモデルをつたためになものです id フィールドだけですが。 ActiveRecord とはなり、ファイルはありません。モデルのすべてのデータベースは、モデルファイルにまれています。

タイムスタンプは、にモデルににまれません。 created_at と updated_at をモデルにするには、

```
include Mongoid::Timestamps
```

のモデルに include Mongoid::Document があります。

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

```
end
```

フィールド

[Mongoidドキュメンテーションによれば](#)、16のなフィールドタイプがあります

- アレイ
- BigDecimal
- ブール
-
-
- く
- ハッシュ
-
- BSON :: ObjectId
- BSON :: バイナリ
-
-
- シンボル
-
- TimeWithZone

フィールドをするには `name` をけて `String` します、これをモデルファイルにします。

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

デフォルトをするには、`default` オプションをします

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

クラシック

Mongoidはな `ActiveRecord` けをにします

- `11` `has_one / belongs_to`
- `1` `has_many / belongs_to`
- `has_and_belongs_to_many`

アソシエーションをするには `User has_many posts` とします、これを `User` モデルファイルにします

```
has_many :posts
```

これを `Post` モデルファイルにします

```
belongs_to :user
```

これにより、Postモデルのuser_idフィールドがされ、Postクラスにuserメソッドがされ、Userクラスにpostsメソッドがされます。

み

Mongoidはみみをします

- embeds_one / embedded_in
- 1 embeds_many / embedded_in

アソシエーションをするにはUser embeds_many アドレスをします、これをあなたのUserファイルにします

```
embeds_many :addresses
```

これをあなたのAddressモデルファイルにAddressます

```
embedded_in :user
```

これはめみますAddress、あなたのUserの、モデルをaddressesあなたにをUserクラス。

データベースびし

MongoidはなときにActiveRecordとたをしようとしています。これらのびしをサポートしています。

```
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
```

オンラインでモンゴイドをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3071/モンゴイド>

55: ルーティング

き

RailsルータはURLをし、コントローラのにそれらをディスパッチします。パスやURLもできるため、ビューのをハードコードするはありません。

「ルーティング」は、に、アプリケーションによってURLがどのように「」されるかをします。Railsのは、にどのコントローラか、そのコントローラ-のどのアクションがのURLをするかです。 Railsアプリケーションでは、ルートは`config/routes.rb`ファイルにかれます。

Examples

リソースルーティング

ルートは`config/routes.rb`されています。それらは、 `resources`または`resource`メソッドをして、しばしばルートのグループとしてされます。

`resources :users`はの7つのルートをしします。すべては`UserController`アクションにマッピングされ、`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'
```

アクションは、の`to`パラメータの#にさ`to`ます。これらのメソッドは、のように`app/controllers/users_controller.rb`でするがあります。

```
class UsersController < ApplicationController
  def index
  end

  def create
  end

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

あなたがされますアクションをすることができます `only`か`except`。

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

ので、のコマンドをすることによって、アプリケーションのすべてのルートができます。

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
```

のコントローラにマップするルートだけをするには

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
```

-gオプションをしてルートができます。これは、ヘルパーメソッド、URLパス、またはHTTPとにするルートをしします。

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
```

さらに、モードで rails サーバをしているときは、`<hostname>/rails/info/routes` からにがするフィルタをして、すべてのルートをするWebページにアクセスできます。それはのようになります

ヘルパー	HTTP	パス	コントローラアクション
パス/URL		[パスマッチ]	
users_path	する	/users(.:format	usersindex
		/users(.:format	userscreate
new_user_path	する	/users/new(.:format	usersnew
edit_user_path	する	/users/:id/edit(.:format	usersedit
user_path	する	/users/:id(.:format	ユーザーshow
	パッチ	/users/:id(.:format	usersupdate
	プット	/users/:id(.:format	usersupdate
		/users/:id(.:format	ユーザーはする

ルートは、メソッドのメンバーのみないコレクションのためにとすることができ resource のわりに、resources して routes.rb。 resource、 index ルートはデフォルトではされませんが、にのようなものをするにります。

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

をしてなをフィルタリングできます。

をするには、のようないくつかのがあります。

- [セグメント](#)、
- [につく](#)
- [な](#)

たとえば、のIPアドレスのみがルートにアクセスすることをするされたベース

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

ののをしてください。 [ActionDispatch :: Routing :: Mapper :: Scoping](#)。

よりなをしたいは、よりなをして、ロジックをラップするクラスをすることができます。

```
# 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
```

1つのフォーム、のボタン

また、フォームのsubmitタグのをとしてして、のアクションにルーティングすることもできます。のボタンがあるフォーム「プレビュー」や「」などがあるは、javascriptをしてフォームの routes.rb URL をするわりに、このを routes.rb にりむことができます。たとえば [commit_param_routing](#) あなたはルールをすることができ submit_tag

Rails submit_tag のパラメータでは、フォームコミットパラメータのをできます

```
# 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
```

スコープのルート

Railsはルートを**する**いくつかの**を**します。

URLによるスコープ

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

これにより、**の**ルートが**され**ます

```
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'
```

サーバーでは、**の**サブフォルダに**いくつかの**ビューを**し**、**ビューとユーザービュー**をするが**に**か
なっています。

スコープモジュール

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

`module`は、**された**のサブフォルダ**の**にあるコントローラファイルを**し**ます。

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

パスヘルパーのプレフィックス**の**をするには、`as`パラメータを**as**ます

```
scope 'admin', as: :administration do
  get 'dashboard'
end

# => administration_dashboard_path
```

Railsは、`namespace`メソッドをして、**の**すべてを**うな**をし**namespace**。 **の**は**です**

```
namespace :admin do
  end

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

コントローラーによるスコープ

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

これによりこれらのルートがされます

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

ネスト

リソースルートは、なりURLをするのにつ`:shallow` オプションをくれます。リソースは1つのレベルのさでネストされるべきではありません。これをする1つのは、いルートをするこです。は、のなないコレクションのURLセグメントかられるこです。にされるネストされたルートは、`:index`、`:create`、および`:new` アクションのみです。りは、のいURLのコンテキストでされます。スコープをカスタマイズするための2つのオプションがあります。

- **shallow_path** されたパラメータでメンバーパスをける

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

- **shallow_prefix** されたパラメータをきヘルパーにする

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

たちはまた、`shallow` ルートをもっとしくすることができます

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

わりにのようにコードされています。

```

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

```

ルートはのとおりです。

		URIパターン
bid_comments	する	/bids/:bid_id/comments(.:format
		/bids/:bid_id/comments(.:format
new_bid_comment	する	/bids/:bid_id/comments/new(.:format
edit_comment	する	/comments/id/edit(.:format
コメント	する	/comments/:id(.:format
	パッチ	/comments/:id(.:format
	プット	/comments/:id(.:format
		/comments/:id(.:format
auction_bids	する	/auctions/:auction_id/bids(.:format
		/auctions/:auction_id/bids(.:format
new_auction_bid	する	/auctions/:auction_id/bids/new(.:format
edit_bid	する	/bids/:id/edit(.:format
	する	/bids/:id(.:format
	パッチ	/bids/:id(.:format
	プット	/bids/:id(.:format
		/bids/:id(.:format
オークション	する	/auctions(.:format
		/auctions(.:format
new_auction	する	/auctions/new(.:format
edit_auction	する	/auctions/id / edit(.:format

		URIパターン
オークション	する	/auctions/:id(.:format
	パッチ	/auctions/:id(.:format
	プット	/auctions/:id(.:format
		/auctions/:id(.:format

くされたルートをする、URLのネストされたは、するデータをするためになとにのみまれています。

ネストされたルートでのりしをけるため、はなリソースをするらしいをします。をするには、メソッドのconcernroutes.rbファイル。このメソッドは、シンボルとブロックをしています。

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

ルートをしていない、このコードではリソースの:concernsをできます。もなはのとおりです。

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

のネストされたリソースはのようになります。

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

これは、えは、のをする。

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

のあることをするには、をするのリソースがです。のリソースは、のいずれかのをしてをびすことができます。

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

リダイレクション

のように、Railsルートでリダイレクトをできます。

4.0

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

4.0

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

すべてののルートがされたパスにリダイレクトすることもできます

4.0

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

4.0

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

メンバールートとコレクションルート

リソースのメンバーブロックをすると、そのリソースベースのルートの々のメンバーにするルートがされます。

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

これにより、のメンバールートがされます。

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

ルートをすると、リソースオブジェクトのコレクションにするルートができます。

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

これにより、のルートがされます。

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

の

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

をつURLパラメータ

IDよりもなurlパラメータをサポートしたいは、にピリオドがまれていると、パーサーにがするがあります。ピリオドにくものは、フォーマットつまり、json、xmlとみなされます。

このをするには、をしてけれなをします。

たとえば、urlのメールアドレスでユーザーレコードをするは、のようにします。

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

ルートルート

rootメソッドをして、ホームページのルートをアプリにすることができます。

```
# 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
```

ターミナルでは、 rake routes Rails 5の rails routes がされます

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

ホームページは、もなルートであり、ルートはされるでけされるため、 rootルートは、ルートファイルののルートにするがあります。

のRESTfulアクション

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

これにより、デフォルトの7つの**RESTful**ルートにえて、のルートがされます。

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

でこれをうには、のようにします。

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

/newパスにアクションをすることもできます

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

どちらがされます

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

あなたの**RESTful**ルートにアクションをするときはしてください。らくのリソースがありません

なロケールの

アプリケーションがなるのでできるは、URLにのロケールがされます。

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

ルートは、`I18n.available_locales`されたロケールをしてアクセスできます。

のアプリケーションをマウントする

`mount`は、のアプリケーションでされるのアプリケーションにラックアプリケーションまたはレールエンジンをマウントするためにされます

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

これで、`route helper` `some_rack_app_path`または`some_rack_app_url`をして、のマウントされたアプリケーションにアクセスできます。

しかし、このヘルパーのをしたい、のようにすることができます

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

これは `myapp_path` と `myapp_url` ヘルパーをし、このマウントされた `app` にナビゲートできます。

リダイレクトとワイルドカードルート

あなたのユーザーになURLをしたいが、にしているのURLにマップしたい。リダイレクトをする

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

まあ、それは良かった。ここでのなは、`#redirect` メソッドをしてのルートに1つのルートをする
ことです。あなたのルートがにシンプルな、それはになです。しかし、のパラメータもしたいは
、`{here}` のパラメータをしてしをするがあります。すべてをむにしてください。

のでは、`as` パラメータでエイリアスをするすることで、のためにルートのをししました。これにより、
`#_path` ヘルパーのようなメソッドでそのをできます。、`$ rake routes` をでテストします。

ルートをのファイルにする

あなたのルートファイルがにきければ、あなたのルートをのファイルにき、Rubyの
`require_relative` メソッドでファイルをインクルードすることができます

`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
```

ネストされたルート

ネストされたルートをするは、`routes.rb` ファイルにのコードを `routes.rb` できます。

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

これにより、のルートがされます。

```

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

```

オンラインでルーティングをむ <https://riptutorial.com/ja/ruby-on-rails/topic/307/ルーティング>

56: レール - エンジン

き

エンジンは、ホストアプリケーションにをすアプリケーションとえることができます。 Railsアプリケーションには「スーパーチャージ」エンジンで、 Rails :: Applicationクラスは Rails :: Engineからくのをしています。

エンジンはなレールアプリケーション/プラグインです。それはのようにく。なエンジンは Device、 Spreeので、レールアプリケーションとにできます。

- rails plugin new [engine name] --mountable

パラメーター

パラメーター	
- マウント	オプションは、 "マウントな"エンジンをすることをジェネレータにします
- フル	オプションは、スケルトンをむエンジンをすることをジェネレータにえます

エンジンは、レールアプリケーションのなプラグインをするためのにいオプションです

Examples

なは

なブログエンジンの

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

なエンジンののは

[デバイス](#) レール

[Spree](#) eコマース

オンラインでレール - エンジンをむ <https://riptutorial.com/ja/ruby-on-rails/topic/10881/レール---エンジン>

57: レールでのい

き

このドキュメントでは、Ruby on Railsでさまざまないをするのをしています。

ここでは、StripeとBraintreeの2つによくられたいプラットフォームについてします。

ドキュメンテーション。

[ストライプ](#)

[ブレンツリー](#)

Examples

ストライプとの

ストライプのをたちのGemfile

```
gem 'stripe'
```

initializers/stripe.rbファイルをします。このファイルには、ストライプアカウントとのになキーがまれています。

```
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]
```

しいをする

```
Stripe::Customer.create({email: email, source: payment_token})
```

このコードは、されたメールアドレスとソースをつStripeにしいをします。

payment_tokenは、クレジットカードやのようないをむクライアントからえられるトークンです。

[Stripe.jsクライアント](#)

ストライプからプランをする

```
Stripe::Plan.retrieve(stripe_plan_id)
```

このコードは、そのIDによってStripeからプランをします。

サブスクリプションの

とプランがあれば、Stripeでいいサブスクリプションをすることができます。

```
Stripe::Subscription.create(customer: customer.id, plan: plan.id)
```

いいサブスクリプションがされ、ユーザーにされます。ユーザーをプランにするときにStripeで
にがくるかを見ることはです。 [ストライプサブスクリプションのライフサイクル](#)については、こ
ちらをご覧ください。

1のいでユーザーにする

には、ユーザーにだけをしたいもあります。

```
Stripe::Charge.create(amount: amount, customer: customer, currency: currency)
```

その、のにして1ユーザーにしています。

なエラー

- はでするがあります。つまり、2000は20になります。 [このをする](#)
- 2つのでユーザーにすることはできません。にユーザーがいつでもEURでされた、ユーザーにUSDをすることはできません。
- ソースなしでユーザーにをすることはできませんおい。

[オンラインでルールでのいをむ](https://riptutorial.com/ja/ruby-on-rails/topic/10929/ルールでのい) <https://riptutorial.com/ja/ruby-on-rails/topic/10929/ルールでのい>

58:

き

、モデル、メソッドなどにしてください。

Examples

リスト

- ADDITIONAL_LOAD_PATHS
- ARGF
- ARGV
- ActionController
- ActionView
- ActiveRecord
- ArgumentError
- アレイ
- BasicSocket
-
- ビンガム
- バインディング
- CGI
- CGIMethods
- CROSS_COMPILING
- クラス
- ClassInheritableAttributes
- の
- ConditionVariable
-
-
- DRb
- DRbIdConv
- DRbObject
- DRbUndumped
- データ
-
-
-
-
- ダイジェスト
- Dir
- ENV
- EOFError

- ERB
-
- エルロー
-
-
- FalseClass
- Fcntl
- ファイル
- FileList
- FileTask
- FileTest
- FileUtils
- フィックスナム
- く
- FloatDomainError
- GC
-
- GetoptLong
- ハッシュ
- IO
- IOError
- IPSocket
- IPソケット
- IndexError
- インフレクター
-
- りみ
- カーネル
- LN_SUPPORTED
- LoadError
- LocalJumpError
- ロガー
- マーシャル
- MatchData
- MatchingData
-
-
- モジュール
- ミューテックス
- MySQL
- MysqlError
- MysqlField
- MysqlRes
- NIL
- NameError
- NilClass

- NoMemoryError
- NoMethodError
- NoWrite
- NotImplementedError
-
- OPT_TABLE
- オブジェクト
- ObjectSpace
- な
-
- PGError
- PGconn
- PGLarge
- PGresult
- プラットフォーム
- PStore
- ParseDate
-
- Proc
- プロセス
- キュー
- RAKEVERSION
-
- ルビー
- RUBY_PLATFORM
- RUBY_RELEASE_DATE
- RUBY_VERSION
- ラック
- レーキ
- RakeApp
- RakeFileUtils
-
- RangeError
- ラショナル
-
- RegexpError
-
- ランタイムエラー
- STDERR
- STDIN
- STDOUT
- ScanError
- ScriptError
- SecurityError
-
- SignalException
- SimpleDelegater

- SimpleDelegator
- シングルトン
- SizedQueue
- ソケット
- SocketError
- StandardError
-
- StringScanner
-
- シンボル
- エラー
- SystemCallError
- SystemExit
- SystemStackError
- TCPServer
- TCPsocket
- TCPserver
- TCPsocket
- TOPLEVEL_BINDING
-
-
- テキスト
-
- ThreadError
- スレッドグループ
-
- トランザクション
- TrueClass
- TypeError
- UDPSocket
- UDPsocket
- UNIXServer
- UNIXSocket
- UNIXサーバー
- UNIXソケット
- UnboundMethod
- URL
- バージョン
-
- YAML
- ZeroDivisionError
- @base_path
- けれる
- アクセス
- Axi
- アクション
-

- アプリケーション2
- りし
- カテゴリー
-
- データベース
- ディスパッチャ
- ディスプレイ1
- ドライブ
- エラー
- フォーマット
- ホスト
- キー
- レイアウト
-
- リンク
- しい
- する
- いた
- パブリック
- もり
- レンダリングする
-
-
-
- する
-
- する
- セッション
- システム
- テンプレート
- テスト
- タイムアウト
- to_s
- タイプ
- URI
-
-

データベースフィールド

- created_at
- created_on
- updated_at
- updated_on
- deleted_at
- パラノイア
-

- lock_version
- タイプ
- id
- {table_name} _count
- ポジション
- parent_id
-
-
- quote_value

Rubyの

- エイリアス
- そして
- ベギン
- ベギン
- ブレーク
-
- クラス
- def
- されている
- う
- else
- elsif
- わり
- わり
- に
-
- にとって
- if
- モジュール
-
- なし
- ない
- または
- やりす
- レスキュー
- リトライ
- リターン
-
- スーパー
- に
-
-
- り
- まで

- いつ
- while
-
- `_ ファイル _`
- `_ LINE _`

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/10818/>

59: テーブル

き

STIは、じモデルからすべてしているのモデルのデータをデータベースののにするというえにづいたパターンです。

Examples

な

まず、データをすするテーブルがです

```
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
```

にいくつかのモデルをさせます

```
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
```

`Guest.create(name: 'Bob')` をすると、ActiveRecordはこれをして、`type: 'Guest'`のUsersテーブルにエントリをします。

レコードをすると、`bob = User.where(name: 'Bob').first`にされるオブジェクトはGuestインスタンスになります。`bob.becomes(User)`つUserとしてにうことができます。

は、サブクラスではなくスーパークラスのパーシャルまたはルート/コントローラをうときにもです。

カスタム

デフォルトでは、STIモデルクラスは`type`というのにされます。しかし、そのは、クラスの`inheritance_column`をオーバーライドすることでできます。えば

```
class User < ActiveRecord::Base
  self.inheritance_column = :entity_type # can be string as well
end

class Admin < User; end
```

こののはのようになります。

```
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
```

`Admin.create`を`Admin.create`、このレコードは、`entity_type = "Admin"` `users`テーブルにされます
`entity_type = "Admin"`

タイプとSTIなしのRailsモデル

STIをびさずにRailsモデルにを`type`すると、`:_type_disabled`を`inheritance_column`りてることでできます。

```
class User < ActiveRecord::Base
  self.inheritance_column = :_type_disabled
end
```

オンラインでテーブルをむ <https://riptutorial.com/ja/ruby-on-rails/topic/9125/テーブル>

60: レールのをってレールとする

Examples

`rails_react gem`をしたRailsのインストール

あなたのGemfileにレールをする

```
gem 'react-rails'
```

インストール

```
bundle install
```

に、インストールスクリプトをします。

```
rails g react:install
```

この

`components.js` マニフェストファイルと `app / assets / javascripts / components /` ディレクトリをします。ここでコンポーネントを `application.js` にします

```
//= require react
//= require react_ujs
//= require components
```

あなたのアプリケーションでの `react_rails` の

React.js ビルド

をすることで、React.jsのビルド、プロダクション、アドオンのにかかわらずをですることができます。デフォルトはのとおりです

```
# 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
```

アドオンをめるには、このをします。

```
MyApp::Application.configure do
  config.react.addons = true # defaults to false
end
```

Railsサーバーをした、// reactをすると、によってされたReact.jsのビルドがされます。

ルールは、React.jsのバージョンとビルドにいくつかのオプションをします。のをやるや、React.jsのコピーをドロップするには、VERSIONS.mdをしてください。

JSX

リアルールをりけたら、サーバーをします。これで、.jsxファイルがアセットパイプラインでされます。

BabelTransformerのオプション

babelのとカスタムプラグインをして、のをしてbabel transpilerにオプションをすことができます

```
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]
}
```

フードのでは、ルールはのためにルビー-バベル-トランスパイラをします。

レンダリングとマウント

react-railsは、ビューヘルパー(react_component)とにならないJavaScriptドライバreact_ujsがまわっています。およびターボリンクをするはターボリンクのマニフェストにUJSドライバがです。

ビューヘルパーは、されたコンポーネントクラスとをつページにdivをします。えば

```
<%= react_component('HelloMessage', name: 'John') %>
<!-- becomes: -->
<div data-react-class="HelloMessage" data-react-
props="{&quot;name&quot;:&quot;John&quot;}"></div>
```

ページがみまわれると、react_ujsドライバはdata-react-classとdata-react-propsをしてページをスキャンし、コンポーネントをマウントします。

Turbolinksがする、コンポーネントはページイベントにマウントされ、ページアンロードにアンマウントされます。Turbolinks> = 2.4.0をします。なぜなら、よりいいイベントをするからです。

Ajaxのびしの、UJSのマウントはjavascriptからびすことでできます

ReactRailsUJS.mountComponentsビューヘルパーのはのとおりで。

```
react_component(component_class_name, props={}, html_options={})
```

`component_class_name`は、グローバルにアクセス可能なコンポーネントクラスの名前を指定します。ドットがある場合があります "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` は `tag`: `data-react-class` と `data-react-props` をめむために `div` の `class`、`id` は、`content_tag` にされます。

`prerender: true` サーバーのコンポーネントをレンダリングする `prerender: true` です。 `**other` その

オンラインでレールの `render_to_string` を読む <https://riptutorial.com/ja/ruby-on-rails/topic/7032/>
レールの `render_to_string` を読む

61:

Examples

コントローラ

コントローラのクラスはになっています。は、コントローラがオブジェクトインスタンスののインスタンスをするためです。

`OrdersController`は`orders`テーブルのコントローラになります。 Railsは`/app/controllers`ディレクトリ`/app/controllers orders_controller.rb`というファイルのクラスをします。

`PostsController`は`posts`テーブルのコントローラになります。

コントローラのクラスにのがある、テーブルはこれらののにアンダースコアがあるものとみなされます。

たとえば、のようにコントローラにがいている`PendingOrdersController`、このコントローラのファイルをします`pending_orders_controller.rb`。

モデル

このモデルは、れることのないMixedCaseのクラスをしてされ、にテーブルのです。

のが`orders`、けられたモデルのは`Order`

のが`posts`、するモデルのは`Post`

Railsは`/app/models`ディレクトリの`order.rb`というファイルでクラスをします。

モデルクラスにののがまれている、テーブルはこれらののにアンダースコアがあるとみなされま

す。モデルのが`BlogPost`、されるテーブルは`blog_posts`になり`blog_posts`。

ビューとレイアウト

コントローラのアクションがレンダリングされると、 Railsはコントローラのにづいて、するレイアウトをつけようとします。

ビューとレイアウトは、 `app/views`ディレクトリにされ`app/views`。

`PeopleController#index`アクションへのリクエストがえられると、 Railsはをします。

- `app/views/layouts/` またはするものが見つからないは`application` の`people`というレイアウト

- デフォルトで `app/views/people/index.html.erb` というビュー
- もしあなたが `index_new.html.erb` という `index_new.html.erb` のファイルをレンダリングしたいのであれば、それを `render 'index_new'` ような `render 'index_new'` アクションで `PeopleController#index`
- `render 'index_new', layout: 'your_layout_name'` することで、すべての `action` にしてなる `layouts` をでき `render 'index_new', layout: 'your_layout_name'`

ファイルとオートローディング

RailsファイルとなRubyファイルは、 `lower_snake_case` ファイルで `lower_snake_case` があります。例えば

```
app/controllers/application_controller.rb
```

`ApplicationController` クラスをむファイルです。 `PascalCase` はクラスとモジュールにされますが、それらがするファイルは `lower_snake_case` なければなり `lower_snake_case`。

Railsはにじてロードファイルをし、 `application_controller` を `ApplicationController` してすなど、さまざまなスタイルでするために "inflection" をするため、したが必要です。

たとえば、 `BlogPost` クラスがしなまだロードされていないとなされた、 `blog_post.rb` というのファイルがされ、そのファイルがロードされます。

したがって、ファイルにをけることもです。ローダーは、ファイルがコンテンツにすることをしているからです。たとえば、 `blog_post.rb` に `just Post` というのクラスがまれているは、 `LoadError` `Expected [some path]/blog_post.rb to define BlogPost`。

`app/something/` えば/ `models / products /` のに `dir` をすると

- しいディレクトリのモジュールとクラスをにしたい、もするはなく、それがみまれます。例えば、 `app/models/products/` you would need to wrap your class in `モジュールProducts`` に `app/models/products/` you would need to wrap your class in。
- しいディレクトリのモジュールとクラスのをしたくないは、 `config.autoload_paths += %W(#{config.root}/app/models/products)` をしてロードする `application.rb` があります。

にがでないには、Railsがでなのをうというにをうがあります。したがって、 "Foot" というモデルがあれば、ルール "" のルーティングおよびそののくのをさせたいは、するコントローラーを " `FootController` " ではなく " `FeetController` " とぶがあります。

コントローラからのモデルクラス

このようにコントローラから `Model` クラスをできます `Context` は `Controller` クラス。

```
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
```

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/1493/>

62: ActiveRecordカラム

- `serialize: <field_plural_symbol>`

Examples

オブジェクトをする

オブジェクトとしてしてデータベースにするのあるがあるは、`serialize`メソッドをしてそののをすると、にされます。

は`text`フィールドとしてするがあります。

モデルでは、フィールドのタイプ `Hash`または`Array` をするがあります。

もっとしくは [serialize >> apidock.com](https://apidock.com)

の

```
class Users < ActiveRecord::Migration[5.0]
  def change
    create_table :users do |t|
      ...
      t.text :preference
      t.text :tag
      ...
      t.timestamps
    end
  end
end
```

あなたのモデルでは

```
class User < ActiveRecord::Base
  serialize :preferences, Hash
  serialize :tags, Array
end
```

オンラインでActiveRecordカラムをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7602/activerecordカラム>

63: なコンスタンティナイズ

Examples

したな

User は ActiveRecord または Mongoid クラスです。 User をプロジェクトの Rails クラスにきえ User Integer や 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]
```

した safe_constantize

これは、されたがプロジェクトのとしてされなためしません。 "array" をしても、ではないためしません。

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

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

オンラインでなコンスタンティナイズをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3015/なコンスタンティナイズ>

64:

Gemfileのドキュメント

がされるプロジェクトのは、GemfileコメントをすることをGemfileます。そうすれば、なでも、がでなくても2にしたとしても、がをするのかはわかります。

これは、のバージョンをしたをえておいて、でバージョンをするのにもちます。

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

Examples

とはですか

は、プログラミングのルビーのプラグインまたはにします。

まさにルールでさえあれば、だけではありません。くのは、ルールやのそれらはのにしているまたはしててられています。

あなたのRailsプロジェクト

Gemfile

あなたのRailsプロジェクトには、Gemfileというファイルがあります。ここでは、プロジェクトにみんでするをできます。すると、bundlerをしてをインストールするがありますBundlerセクションを。

Gemfile.lock

これをませたら、あなたのGemfile.lockはしくされたとそのでされます。このファイルは、しているをロックして、そのファイルでされているのバージョンをするようにします。

```
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)
```

これは、`devise` です。 `Gemfile.lock`、バージョン4.0.3がされ、されたバージョンをするのマシンまたは `Gemfile.lock` サーバーにプロジェクトをインストールするときにされます。

1の、グループ、またはコミュニティがをい、しています。したは、の `issues` がされた、または `features` がされたにリリースされます。

、リリースは [Semantic Versioning 2.0.0](#) のにいます。

バンドラー

をい、するものは、 `bundler` をうことです。 [Bundler](#) は、 [Bower](#) にするパッケージマネージャです。

バンドラーをするには、まずそれをインストールするがあります。

```
gem install bundler
```

`Gemfile` してする `Gemfile` で、 `Gemfile` をしてするだけです

```
bundle
```

あなたので。これにより、しくされたがプロジェクトにインストールされます。がしたは、にプロンプトがされます。

よりながなは、 [ドキュメント](#) をご覧ください。

Gemfiles

まず、 `gemfiles` には、 `RubyGems` サーバーのURLのでなくとも1つのソースがです。

`bundle init` して、デフォルトの `rubygems.org` ソースをつ `Gemfile` をし `bundle init` 。 `https` をすると、サーバーへののが `SSL` でされます。

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

に、なバージョンをむをします。

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

`>= 1.0` のようなほとんどのバージョンはです。はなをちます。 `2.0.3` は `> 2.0.3` および `< 2.1` とじです。 `2.1` は `> 2.1` および `< 3.0` とである。 `2.2.beta` は、 `2.2.beta.12` のようなプレリリースとする。

`Git` リポジトリは、リポジトリに1つまたはのながまれているり、なでもあります。 `:tag`、`:branch`

、または`:ref`をしてチェックアウトするものをします。デフォルトは`master`ブランチです。

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

ファイルシステムからアンパックされたgemをしたいは、`path`オプションをgemのファイルをむパスにするだけです。

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

はグループにできます。グループは、インストールに`--without`をしてすることも、にすべてをとすることもできます `Bundler.require`を。

```
gem 'rails_12factor', group: :production

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

あなたはとGemfileにルビーのなバージョンをすることができ`ruby`。GemfileがのRubyバージョンにロードされている、Bundlerはきでをさせます。

```
ruby '2.3.1'
```

RVM(Ruby Version Manager)をしている`gemset`、プロジェクトに`gemset`をする`gemset`をおめします。`gemset`は、をいからするためうことができます。プロジェクトごとに`gemset`すると、のすべてのプロジェクトをすことなく、あるプロジェクトのおよびのバージョンをすることができます。プロジェクトでは、のだけをするがあります。

RVMは、ルビインタプリタごとに`@global gemset`します`>= 0.1.8`。えられたルビーの`@global gemset`インストールするは、そのルビとしてしたのすべてのセットでできます。これは、すべてのプロジェクトがのRubyインタプリタインストールにインストールされたじをできるようにするためのいです。

セットの

すでに`ruby-2.3.1`インストールされており、のコマンドをしてしたとします。

```
rvm use ruby-2.3.1
```

このルビーバージョンの`gemset`をする

```
rvm gemset create new_gemset
```

`new_gemset`は`new_gemset`のです。Rubyでなのリストをるには

```
rvm gemset list
```

すべてのルビーバージョンのをリストする

```
rvm gemset list_all
```

リストからgemsetをする `new_gemset` はしたいgemsetであるとします。

```
rvm gemset use new_gemset
```

のルビーバージョンにしたいは、gemsetでルビバージョンをすることもできます

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

のルビーバージョンのデフォルトのgemsetをする

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

インストールされているすべてのをジェムセットからりくには、のでにすることができます

```
rvm gemset empty new_gemset
```

1つのルビーからのルビーにgemsetをコピーするには、のようになります。

```
rvm gemset copy 2.1.1@rails4 2.1.2@rails4
```

gemsetをするには

```
rvm gemset delete new_gemset
```

のgemsetをするには

```
rvm gemset name
```

グローバルgemsetにをインストールするには

```
rvm @global do gem install ...
```

Ruby インストールの**Gemset** セットの

しいルビーをインストールすると、RVMは2つのジェムセットデフォルトのジェムセットとグローバルジェムセットをするだけでなく、ユーザーファイルのセットをしてインストールするをします。

`~/.rvm/gemsets` であると、rvmはインストールされているルビについてツリーをって `global.gems` と

default.gems をします。 ree-1.8.7-p2010.02 のをして、 rvm はのファイルをチェックしますそしてそこからインポートします

```
~/ .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
```

たとえば、 ~/ .rvm/gemsets/global.gems をの2をしてしたとします。

```
bundler
awesome_print
```

あなたがしいルビーをインストールするたびに、これらの2つのがグローバルジェムセットにインストールされます。 default.gems および global.gems ファイルは、 rvm のにきされます。

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3130/>

65: のの

Examples

の

とメールをつActiveRecord Userクラスをっている、FactoryGirlにさせることで、そのファクトリをできます。

```
FactoryGirl.define do
  factory :user do # it will guess the User class
    name      "John"
    email     "john@example.com"
  end
end
```

あるいは、にしたり、をしたりすることもできます

```
FactoryGirl.define do
  factory :user_jack, class: User do
    name      "Jack"
    email     "jack@example.com"
  end
end
```

あなたのでは、FactoryGirlのメソッドをのようになうことができます

```
# 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)
```

もなはのとおりです。

```
# 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)
```

オンラインでののをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8330/のの>

66:

Examples

インストールとテスト

ローカルのためにまずやりたいことは、あなたのマシンにElasticSearchをインストールし、それがしているかどうかをテストすることです。Javaをインストールするがあります。インストールはかなりです

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

にします

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

それをテストするために、もなは`curl`です。にはかかることがあるので、にがないはてないでください。

```
curl localhost:9200
```

```
{
  "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"
}
```

のためのツールの

ElasticSearchESをいめるときには、データをするのにグラフィカルなツールをしておくといでしょう。 `elasticsearch-head` とばれるプラグインはこれだけです。インストールするには、のをします。

- どのフォルダESがインストールされているか `ls -l $(which elasticsearch) ls -l $(which elasticsearch)`
- このフォルダに`cd`して、プラグインのインストールバイナリをします `elasticsearch/bin/plugin -install mobz/elasticsearch-head`
- ブラウザで `http://localhost:9200/_plugin/head/` をき `http://localhost:9200/_plugin/head/`

すべてがどおりにしていれば、あなたはのデータをできるなGUIをているはずです。

き

ElasticSearchにはされたJSON APIがありますが、おそらくあなたのためにそれをういくつかのライブラリをしたいとうでしょう

- [Elasticsearch](#) - HTTP APIのレベルラッパー
- [Elasticsearch-rails](#) - ActiveRecordまたはRepositoryパターンのいずれかをしてRailsモデルとElasticSearchをするのにつ、なRails
- [Chewy](#) - にしており、いなくよりいをつけている

をテストするためののオプションをしましょう

```
gem install elasticsearch
```

その、ルビーをしてしてみてください

```
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

すぐにelasticsearchをセットアップしたいは、searchkick gemをすることができます

```
gem 'searchkick'
```

したいモデルにsearchkickをしてください。

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

インデックスにデータをします。

```
Product.reindex
```

クエリをするには、をします。

```
products = Product.search "apples"
```

```
products.each do |product|  
  puts product.name  
end
```

かなりい、elasticsearchのはとされません;-)

はこちら <https://github.com/ankane/searchkick>

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/6500/>

67:

Examples

カスタム

`config/`ディレクトリにYAMLファイルをします。 `config/neo4j.yml`

`neo4j.yml`のはのようになりますにするため、 `default`はすべてのでされています。

```
default: &default
  host: localhost
  port: 7474
  username: neo4j
  password: root

development:
  <<: *default

test:
  <<: *default

production:
  <<: *default
```

`config/application.rb`

```
module MyApp
  class Application < Rails::Application
    config.neo4j = config_for(:neo4j)
  end
end
```

さて、あなたのカスタムはのようアクセスです

```
Rails.configuration.neo4j['host']
#=> localhost
Rails.configuration.neo4j['port']
#=> 7474
```

RailsのAPIドキュメントでは、 `config_for`メソッドについてのようになっています。

のRailsで`config / foo.yml`をみむのにです。

`yaml` ファイルをしたくない

のコードをRailsオブジェクトにするには、 `config.x`プロパティのでカスタムをい`config.x`。

```
config.x.payment_processing.schedule = :daily
config.x.payment_processing.retries  = 3
config.x.super_debugger = true
```

これらのポイントは、オブジェクトをじてできます。

```
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
```

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2558/>

68:

Examples

Railsの

レールのファイルは `config/environments/` ます。デフォルトでは、レールには3つの、 `development`、`production`、`test` ます。ファイルをすることによって、そのののみをしています。

Railsは `config/application.rb` もファイルをとっています。これはのファイルです。ここでされたは、でされたによってきされるためです。

`Rails.application.configure do` ブロックのオプションをまたはし、オプションは `config.` まります `config.`

データベース

railsプロジェクトのデータベースは `config/database.yml` ファイルにあり `config/database.yml`。
`rails new` コマンドをしてプロジェクトをし、するデータベースエンジンをしない、railsは `sqlite` をデフォルトデータベースとしてします。デフォルトのな `database.yml` ファイルは、のようになりま
す。

```
# SQLite version 3.x
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3

production:
  <<: *default
  database: db/production.sqlite3
```

しいプロジェクトをしているときにデフォルトのデータベースをしたい、データベースをすることができ `rails new hello_world --database=mysql`

Railsのな

Rails::Railtieオブジェクトにしてのオプションをびすがあります

- **config.after_initialize** レールがアプリケーションをしたにされるブロックをとります。
- **config.asset_host** アセットのホストをします。これは、コンテンツネットワークをするにです。これは、`config.action_controller.asset_host`です `config.action_controller.asset_host`
- **config.autoload_once_paths** このオプションは、Railsがをみみするパスのを付けれます。デフォルトはのです
- **config.autoload_paths** これは、Railsがをみみするパスのを付けれます。デフォルトでは、`app`にあるすべてのディレクトリ
- **config.cache_classes** リクエストごとにクラスとモジュールをリロードするかどうかをします。モードでは、これはデフォルトで`false`なり、モードとテストモードではデフォルトで`true`になり`true`
- **config.action_view.cache_template_loading** これは、リクエストでテンプレートをリロードするかどうかをします。デフォルトでは、`config.cache_classes`になります
- **config.beginning_of_week** デフォルトののをします。なのの `:monday` が必要です。
- **config.cache_store** するキャッシュストアをします。オプションには、`:file_store`、`:memory_store`、`mem_cache_store`または`null_store`ます。
- **config.colorize_logging** ロギングをけするかどうかをします。
- **config.eager_load** すべてのみの`eager -load`
- **config.encoding** アプリケーションのエンコーディングをします。デフォルトは`UTF-8`
- **config.log_level** Rails Loggerのをします。デフォルトでは、すべてので`:debug`れます。
- **config.middleware** これをしてアプリケーションのミドルウェアをする
- **config.time_zone** これは、アプリケーションのデフォルトのタイムゾーンをします。

アセットの

アセットのには、のオプションをできます

- **config.assets.enabled** アセットパイプラインをにするかどうかをします。デフォルトは`true`です。
- **config.assets.raise_runtime_errors** これにより、ランタイムエラーチェックがになります
 - `development` `modedevelopment` `mode`
- **config.assets.compress** アセットをします。プロダクションモードでは、これはデフォルトで`true`になります
- **config.assets.js_compressor** するJSコンプレッサーをします。オプションには、`:closure`、`:uglifyer`、`:yui`
- **config.assets.paths** アセットをするパスをします。
- **config.assets.precompile** `rake assets:precompile`がされたときに、プリコンパイルされるアセットをできます。
- **config.assets.digest** このオプションでは、に`MD-5`をできます。モードでは、デフォルトで`true`になります。

- **config.assets.compile** プロダクションモードでのライブ `Sprockets` コンパイルをりえます。

ジェネレータの

Railsでは、`rails generate` コマンドを `rails generate` ときにされるジェネレータをできます。このメソッドは、`config.generators` ブロックを

```
config.generators do |g|
  g.orm :active_record
  g.test_framework :test_unit
end
```

いくつかのオプションがあります

オプション		デフォルト
	をするときアセットをする	
<code>force_plural</code>	のモデルが	
ヘルパー	ヘルパーをするかどうかをします。	
<code>integration_tool</code>	ツールをする	<code>test_unit</code>
<code>javascript_engine</code>	JSエンジンをする	<code>:js</code>
<code>resource_route</code>	リソースルートをする	
<code>stylesheet_engine</code>	スタイルシートエンジンをする	<code>:cs</code>
<code>scaffold_stylesheet</code>	スキヤフオールディングにCSSをする	
<code>test_framework</code>	テストフレームワークの	<code>Minitest</code>
<code>template_engine</code>	テンプレートエンジンをする	<code>:erb</code>

オンラインでをむ <https://riptutorial.com/ja/ruby-on-rails/topic/2841/>

69: いルーティング

Examples

1. いの

ディープネストをするの1つは、のにスコープされたコレクションアクションをして、のをるが、メンバーのアクションをネストしないことです。いえれば、ので、リソースをにするルートをするだけです。

```
resources :articles, shallow: true do
  resources :comments
  resources :quotes
  resources :drafts
end
```

DSLのいは、すべてのネスティングがいスコープをします。のとじルートがされます。

```
shallow do
  resources :articles do
    resources :comments
    resources :quotes
    resources :drafts
  end
end
```

いルートのカスタマイズするためのスコープには2つのオプションがあります。 `shallow_path`はされたパラメータでメンバーパスのにけます

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

Rakeコマンドをして、にするようにされたルートをしします。

```
rake routes
```

オンラインでいルーティングをむ <https://riptutorial.com/ja/ruby-on-rails/topic/7775/いルーティング>

70: のフォルダからCSVファイルをインポートする

き

ここでは、フォルダにのCSVファイルがあるとします。CSVファイルは、コンソールからデータベースをアップロードしてコマンドをきむがあります。またはのプロジェクトでのコマンドをし、このモデルをします。

Examples

コンソールコマンドからCSVをアップロードする

ターミナルコマンド

```
rails g model Product name:string quantity:integer price:decimal{12,2}
rake db:migrate
```

Latesはコントローラをします。

ターミナルコマンド

```
rails g controller Products
```

コントローラコード

```
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
```

モデル

```
class Product < ApplicationRecord
  def self.import(file)
    CSV.foreach(file.path, headers: true) do |row|
      Product.create! row.to_hash
    end
  end
end
```

routes.rb

```
resources :products
```

app / config / application.rb

```
require 'csv'
```

console `いて` run

```
=> ProductsController.new.create #=> Uploads your whole CSV files from your folder directory
```

オンラインでのフォルダからCSVファイルをインポートするをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8658>のフォルダからcsvファイルをインポートする

71: パネルの

き

パネルをレールアプリケーションにするは、ほんのです。

1. Open gemファイルとライターの「rails_admin」、「1.0」
2. バンドルインストール
3. rails g rails_admininstall
4. デフォルトのEnterキーをすは、のルートについてねられます。
5. はapp / config / initializers / rails_admin.rbにき、のコードをりけてください
config.authorize_with redirect_to main_app.root_path current_user.tryadminendこのコード
では、だけがyoursite.com/adminそのにアクセスできますユーザーはrootpathにリダイレク
トされます。
6. については、こののドキュメントをチェックしてください。
https://github.com/sferik/rails_admin/wiki

それをとしないは、あなたのウェブサイトのをちたいにしてください。これはactive_adminのよ
りもです。これは、ユーザーのどのでもできます。また、4にユーザーをうことをれないでくだ
さい。ロールをするにはcancanをします。

Examples

rails_admin gemをしたパネルからのスクリーンショットはほとんどありません。

あなたがることができるように、このレイアウトはにキャッチし、ユーザーフレンドリーです
。

NAVIGATION

[Blogs](#)

[Users](#)

Site Administration

Dashboard

 Dashboard

Model name	Last created	Records
Blogs	about 7 hours ago	
Users	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

オンラインでパネルのをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8128/パネルの>

72: キーのな

き

くのサードパーティAPIでは、をぐためのキーがです。らがあなたにをした、があなたのをむことができるように、をリポジトリにコミットしないことがにです。

Examples

フィガロとキーの

あなたのGemfileに`gem 'figaro' figaro gem 'figaro'`をし、`bundle install`をしてください。に、`bundle exec figaro install`します`bundle exec figaro install; config / application.yml`がされ、`.gitignore`ファイルにされ、バージョンコントロールにされなくなります。

あなたのキーは、ので`application.yml`にすることができます

```
SECRET_NAME: secret_value
```

`SECRET_NAME`と`secret_value`はAPIキーのとです。

また、`config / secrets.yml`にこれらののをけるがあります。それぞれのでなるをつことができます。ファイルはのようになります。

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

これらのキーのいはさまざまですが、たとえばの`some_component`が`secret_name`アクセスするがあるとします。`config / environments / development.rb`には、のようになります。

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

に、Herokuのプロダクションをスピンアップさせたいとしましょう。このコマンドは、`config / environments / production.rb`のをHerokuにアップロードします

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

オンラインでキーの名をむ <https://riptutorial.com/ja/ruby-on-rails/topic/9711/キーの名>

73: パイプライン

き

アセットパイプラインは、JavaScriptとCSSアセットをしたり、したり、したりするためのフレームワークをします。また、これらのをCoffeeScript、Sass、ERBなどののやプリプロセッサですることもできます。これにより、アプリケーションのアセットをののアセットとにすることができます。たとえば、jquery-railsにはjquery.jsのコピーがまれ、RailsにAJAXがになっています。

Examples

レイクタスク

デフォルトでは、`sprockets-rails`はのレーキタスクがまれています。

- `assets:clean[keep]` いコンパイルみのをする
- `assets:clobber` コンパイルされたアセットをする
- `assets:environment` のコンパイルをロードする
- `assets:precompile` `config.assets.precompile` `config.assets.precompile` で `config.assets.precompile` されたすべてのアセットをコンパイルし `config.assets.precompile`

マニフェストファイルとディレクティブ

`assets config/initializers/assets.rb` `config/initializers/assets.rb` には、プリコンパイルするよようににされたいくつかのファイルがあります。

```
# Precompile additional assets.
# application.coffee, application.scss, and all non-JS/CSS in app/assets folder are already
added.
# Rails.application.config.assets.precompile += %w( search.js )
```

このでは、`application.coffee`と`application.scss`は「マニフェストファイル」とばれています。このファイルは、のJavaScriptまたはCSSアセットをめるためにするがあります。のコマンドをできます。

- `require <path>` Rubyの`require`の`require`ディレクティブ。パスのファイルにをするをし、ソースファイルのにだけロードされるようにします。
- `require_directory <path>` のディレクトリにすべてのファイルがです。これは、ネストされたディレクトリにわないたため、`path/*`とてい`path/*`。
- `require_tree <path>` ディレクトリのすべてのネストされたファイルがです。そのグロブにするのは`path/**/*`です。
- `require_self` の`require`ディレクティブのに、のファイルのをします。インデックスファイルにのがロードされるにするのあるグローバルスタイルをむのがなCSSファイルです。

- stub <path> ファイルをインクルードからする
- depend_on <path> をファイルにめることなくを depend_on <path> できます。これはキャッシュでされます。ファイルにえられたすべての、ソースファイルのキャッシュをにします。

application.scss ファイルはのようになります。

```
/*
 *= require bootstrap
 *= require_directory .
 *= require_self
 */
```

もう1つのは application.coffee ファイルです。ここで jquery と Turbolinks をめて

```
#= require jquery2
#= require jquery_ujs
#= require turbolinks
#= require_tree .
```

CoffeeScript をせずに JavaScript をする、はのようになります。

```
//= require jquery2
//= require jquery_ujs
//= require turbolinks
//= require_tree .
```

な

アセットパイプラインをするなは2つあります。

1. モードでサーバーをすると、サーバーはにし、をオンザフライでします。
2. プロダクションモードでは、おそらくこれをとってアセットを、バージョン、、コンパイルします。のコマンドをして、これをうことができます。

```
bundle exec rake assets:precompile
```

オンラインでパイプラインをむ <https://riptutorial.com/ja/ruby-on-rails/topic/3386/パイプライン>

74: にわたる Rails フレームワーク

き

Railsにれていなくても、のRailsアプリケーションです、どのフレームワークがいつされたのかをするのはらわしいかもしれません。このトピックは、Railsバージョンのすべてのフレームワークのなりリストになるようにされています。

Examples

のバージョンの**Rails**でなフレームワークをつける

```
config.frameworks
```

オプションをして、フレームワークをす `Symbol` のをします。

Rails 1.xのRailsバージョン

- ActionMailer
- ActionPack
- ActionWebService
- ActiveRecord
- ActiveSupport
- すり

Rails 2.xのRailsフレームワーク

- ActionMailer
 - ActionPack
 - ActiveRecord
 - ActiveSupport
 - すり
- `ActiveResource` `ActiveWebService`は `ActiveResource` にきえられました。*Rails*はデフォルトで*Rails*からRESTにしました

Rails 3.xのRailsフレームワーク

- ActionMailer
- ActionPack
- ActiveSupport
- すり

すり

オンラインでにわたるRailsフレームワークをむ <https://riptutorial.com/ja/ruby-on-rails/topic/8107/>
にわたるrailsフレームワーク

クレジット

S. No		Contributors
1	Ruby on Railsスタートガイド	Abhishek Jain , Adam Lassek , Ajay Barot , animuson , ArtOfCode , Aswathy , Community , Darpan Chhatravala , Darshan Patel , Deepak Mahakale , fybw id , Geoffroy , hschin , hvenables , Jon Wood , kfrz , Kirti Thorat , Lorenzo Baracchi , Luka Kerr , MauroPorrasP , michaelpri , nifCody , Niyanta , olive_tree , RADan , RareFever , Richard Hamilton , sa77 , saadlulu , sahil , Sathishkumar Jayaraj , Simone Carletti , Stanislav Valášek , theoretisch , tpei , Undo , uzaif , Yana
2	ActionCable	Ich , Sladey , Undo
3	ActionController	Adam Lassek , Atul Khanduri , Deep , Fire-Dragon-DoL , Francesco Lupo Renzi , jackerman09 , RamenChef , Sven Reuter
4	ActionMailer	Adam Lassek , Atul Khanduri , jackerman09 , owahab , Phil Ross , Richard Hamilton , Rodrigo Argumedo , William Romero
5	ActiveModel	Adam Lassek , RamenChef
6	ActiveRecord	Adam Lassek , AnoE , Bijal Gajjar , br3nt , D-side , Francesco Lupo Renzi , glapworth , jeffdill2 , Joel Drapper , Luka Kerr , maartenvanvliet , marcamillion , Mario Uher , powerup7 , Sebastialonso , Simone Carletti , Sven Reuter , walid
7	ActiveRecordトランザクション	abhas , Adam Lassek
8	ActiveRecordのロック	Adam Lassek , fatfrog , Muaaz Rafi
9	ActiveRecordの	Adam Lassek , Aigars Cibulskis , Alex Kitchens , buren , Deepak Mahakale , Dharam , DSimon , Francesco Lupo Renzi , ginioux , Hardik Kanjariya ツ , hschin , jeffdill2 , Kirti Thorat , KULKING , maartenvanvliet , Manish Agarwal , Milo P , Mohamad , MZaragoza , nomatteus , Reboot , Richard Hamilton , rii , Robin , Rodrigo Argumedo , rony36 , Rory O'Kane , tessi , uzaif , webster
10	ActiveRecord	Adam Lassek , Colin Herzog , Deepak Mahakale , dgilperez , dodo121 , ginioux , Hai Pandu , Hardik Upadhyay , mmichael , Muhammad Abdullah , pablofullana , Richard Hamilton

11	ActiveRecordインターフェース	Adam Lassek , Ajay Barot , Avdept , br3nt , dnsh , Fabio Ros , Francesco Lupo Renzi , giniouxe , jeffdill2 , MikeAndr , Muhammad Abdullah , Niyanta , powerup7 , rdnewman , Reboot , Robin , sa77 , Vishal Taj PM
12	ActiveSupport	Adam Lassek
13	CanCanによる	4444 , Ahsan Mahmood , dgilperez , mlabarca , toobulkeh
14	DeviseをしてApiをする	Vishal Taj PM
15	GoogleMapsとRailsの	fiedl
16	HerokuにRailsアプリケーションをデプロイする	B Liu , hschin
17	l18n -	Cyril Duchon-Doris , Francesco Lupo Renzi , Frederik Spang , gwcodes , Jorge Najera T , Lahiru , RamenChef
18	Prawn PDF	Awais Shafqat
19	Rails 5	thiago araujo
20	Rails 5 APIのオートメーション	HParker
21	Rails API	Adam Lassek , hschin
22	Rails Cookbook - なレールレシピ/とコーディングテクニック	Milind
23	railsアプリケーションにAmazon RDSをする	Sathishkumar Jayaraj
24	Railsアプリケーションのテスト	HParker
25	Railsエンジン - モジュールレール	Mayur Shah
26	Railsでのユーザ	Abhinay , Ahsan Mahmood , Antarr Byrd , ArtOfCode , dgilperez , Kieran Andrews , Luka Kerr , Qchmqz , uzaif ,

27	Railsでをする	B8vrede , Rory O'Kane , Umar Khan
28	Railsのアップグレード	hschin , michaelpri , Rodrigo Argumedo
29	Railsはコマンドをする	Adam Lassek , ann , Deepak Mahakale , Dharam , Hardik Upadhyay , jackerman09 , Jeremy Green , marcamillion , Milind , Muhammad Abdullah , nomatteus , powerup7 , Reub , Richard Hamilton
30	Railsベストプラクティス	Adam Lassek , Brandon Williams , Gaston , giniouxe , Hardik Upadhyay , inye , Joel Drapper , Josh Caswell , Luka Kerr , ma_il , msohng , Muaaz Rafi , piton4eg , powerup7 , rony36 , Sri , Tom Lazar
31	Railsロガー	Alejandro Montilla , hgsongra
32	RSpecとRuby on Rails	Ashish Bista , Scott Matthewman , Simone Carletti
33	Ruby on Railsのコードとクリーンアップのためのツール	Akshay Borade
34	Ruby on Railsのネストされたフォーム	Arslan Ali
35	アクティブジョブ	Brian , owahab
36	アクティブなジョブ	tirdadc
37	アクティブモデルシリアライザ	Flip , owahab
38	アクティブレコード	giniouxe , Hardik Upadhyay , Khanh Pham , Luka Kerr , Manish Agarwal , Niyanta , RareFever , Raynor Kuang , Sapna Jindal
39	キャッシング	ArtOfCode , Cuisine Hacker , Khanh Pham , RamenChef , tirdadc
40	クラス	Deep , hadees , HParker
41	ターボリンク	Mark
42	デコレータパターン	Adam Lassek
43	デバッグ	Adam Lassek , Dénes Papp , Dharam , Kelseydh , sa77 , titan
44	デフォルトのRailsアプリケーションをす	Whitecat

	る	
45	デフォルトのタイムゾーンをする	Mihai-Andrei Dinculescu
46	ドッカーのルール	ppascualv , Sathishkumar Jayaraj
47	ハイパーループをつたReact.jsとRailsの	Mitch VanDuyn
48	ビュー	danirod , dgilperez , elasticman , Luka Kerr , MikeC , MMachinegun , Pragash , RareFever
49	ファイルのアップロード	Sergey Khmelevskoy
50	フォームヘルパー	aisflat439 , owahab , Richard Hamilton , Simon Tsang , Slava.K
51	フレンドリーなID	Thang Le Sy
52	モデルAASM	Lomefin
53	モンゴイド	Ryan K , tes
54	ルーティング	Adam Lassek , advishnuprasad , Ahsan Mahmood , Alejandro Babio , Andy Gauge , AppleDash , ArtOfCode , Baldrick , cl3m , Cyril Duchon-Doris , Deepak Mahakale , Dharam , Eliot Sykes , esthervillars , Fabio Ros , Fire-Dragon-DoL , Francesco Lupo Renzi , giniouxe , Giuseppe , Hassan Akram , Hizqeel , HungryCoder , jkdev , John Slegers , Jon Wood , Kevin Sylvestre , Kieran Andrews , Kirti Thorat , KULKING , Leito , Mario Uher , Milind , Muhammad Faisal Iqbal , niklashultstrom , nuclearpidgeon , pastullo , Rahul Singh , rap-2-h , Raynor Kuang , Richard Hamilton , Robin , rogerdpack , Rory O'Kane , Ryan Hilbert , Ryan K , Silviu Simeria , Simone Carletti , sohail khalil , Stephen Leppik , TheChamp , Thor odinson , Undo , Zoran ,
55	ルール - エンジン	Deepak Kabbur
56	ルールでのい	ppascualv , Sathishkumar Jayaraj
57		Emre Kurt
58	テーブル	Niyanta , Ruslan , Slava.K , toobulkeh , Vishal Taj PM
59	ルールのをってルールとする	Kimmo Hintikka , tirdadc
60		Andrey Deineko , Atul Khanduri , br3nt , Flambino , giniouxe ,

		hgsongra , Luka Kerr , Marko Kacanski , Muhammad Abdullah , Sven Reuter , Xinyang Li
61	ActiveRecordカラム	Fabio Ros
62	なコンスタンティナイズ	Eric Bouchut , Ryan K
63		Deep , hschin , ma_il , MMachinegun , RamenChef
64	のの	Rafael Costa
65		Don Giovanni , Luc Boissaye
66		Ali MasudianPour , Undo
67	いルーティング	Darpan Chhatravala
68	のフォルダからCSV ファイルをインポートする	fool
69	パネルの	Ahsan Mahmood , MSathieu
70	キーのな	DawnPaladin
71	パイプライン	fybw id , Robin
72	にわたるRailsフレームワーク	Shivasubramanian A