小学生でもわかる Ruby on Rails 入門　（ https://openbook4.me/projects/92/ ）

Rails は Web アプリケーションフレームワークで、Ruby 用のパッケージ管理システム RubyGems の一つとして公開されている。（gem：宝石）Rails の導入事例：Twitter、Github、CookPad

－１．Ruby をインストール

０．Rails のインストール：　**gem install rails**

１．Rails のひな形生成：　rails new mitakalab-twitter

- Gemfile
- app
  - | - assets
  - | - **controllers**
  - | - helpers
  - | - models
  - | - **views**
- config
  - | - **routes.rb**
- db

Web アプリケーションサーバ Rails の動作
ユーザーが URL でアクセス
http://0.0.0.0:3000/users/show/ryooopan
⇒ ルーティングが仕分け
⇒ コントローラが値を入れる
⇒ ビューがコントローラから渡された値を表示

２．Controller と View 生成：　rails g controller users index show　　注：erb：Embed RuBy

app/controllers/**users**_controller.rb
app/views/**users/index**.html.erb
app/views/**users/show**.html.erb

View: HTML template
Controller: View に値を渡す

db/seeds.rb を編集

```
@user = User.new
@user.name = 'Ryo Suzuki'
@user.username = 'ryooopan'
@user.location = 'Kanagawa, Japan'
@user.about = 'Hello, I am Ryo. I am from database!'
@user.save
```

３．Data base 生成：rake db:create
db/development.sqlite3

rails g model user name:string **username**:string **location**:string **about**:text
app/models/user.rb　　モデルファイル
db/migrate/20130630062417_create_users.rb　　マイグレーションファイル

rake db:migrate　　データベース生成
db/seeds.rb　　初期データを編集
rake db:seed　　データベースに初期データ投入

db/migrate/20130630062417_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :username
      t.string :location
      t.text :about

      t.timestamps
    end
  end
end
```

４．Rails 起動　**rails s**　　**注：s　server**

http://0.0.0.0:3000/users/show/ryooopan

Web アプリケーションサーバ Rails の動作

ユーザーが URL でアクセス
⇒ ルーティングが仕分け
⇒ コントローラが値を入れる
⇒ ビューがコントローラから渡された値を表示

http://0.0.0.0:3000/users/show/ryooopan

```
- Gemfile
        - app
            |- assets
            |- controllers
            |- helpers
            |- models
            |- views
        - config
            |- routes.rb
        - db
```

MitakalabTwitter::
::の左は module 名

config/routes.rb

```
MitakalabTwitter::Application.routes.draw do
  get "users/index"
  get "users/show/:username" => "users#show"
  # The priority is based upon order of creation: first created -> highest priority.
  # See how all your routes lay out with "rake routes".
  ...
end
```

app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def index
  end

  def show
    @user = User.find_by(:username =>
                params[:username])
  end

end
```

Railsは URL の最後の値を params[]で受け取る

@user　インスタンス変数

app/views/users/show.html.erb　編集

```
<h1><%= @user[:name] %></h1>
<p><%= @user[:username] %></p>
<ul>
  <li>Location : <%= @user[:location] %></li>
  <li>About      : <%= @user[:about] %></li>
</ul>
```

db/migrate/20130630062417_create_users.rb

```
class CreateUsers < ActiveRecord::Migration
  def change
    create_table :users do |t|
      t.string :name
      t.string :username
      t.string :location
      t.text :about
      t.timestamps
    end
  end
end
```

db/seeds.rb　編集

```
@user = User.new
@user.name = 'Ryo Suzuki'
@user.username = 'ryooopan'
@user.location = 'Kanagawa, Japan'
@user.about = 'Hello, I am Ryo. I am from database!'
@user.save
```

ツイートを保存するためのビュー、コントローラ、モデルを作る
ビューとコントローラを生成

```
rails g controller tweets index show new
```

モデルを生成

```
rails g model tweet title:string content:text
```

データベースに反映

```
rake db:migrate
```

ツイートを入力するためのフォームを作る

app/views/tweets/new.html.erb　　http://0.0.0.0:3000/tweets/new

```
<%= form_for Tweet.new do |f| %>
  <%= f.label :title %>
  <%= f.text_field :title %>
  <%= f.label :content %>
  <%= f.text_area :content %>
  <%= f.submit %>
<% end %>
```

config/routes.rb を編集

```
MitakalabTwitter::Application.routes.draw do
  get "tweets/index"
  get "tweets/show"
  get "tweets/new"
  post "tweets" => "tweets#create"

  get "users/index"
  get "users/show/:username" => "users#show"
  ...
end
```

params = { :tweet => { :title => "This is first tweet title", :content => "I am making how to make twitter app." } }
params[:tweet][:title]　　params[:tweet][:content]

app/controllers/tweets_controller.rb

```
def create
  @tweet = Tweet.new
  @tweet.title = params[:tweet][:title]
  @tweet.content = params[:tweet][:content]
  @tweet.save
end
```

app/controllers/tweets_controller.rb

```
def create
  @tweet = Tweet.new
  @tweet.title = params[:tweet][:title]
  @tweet.content = params[:tweet][:content]
  @tweet.save
  redirect_to '/tweets/index'
end
```

app/views/tweets/index.html.erb

```
<% @tweets.each do |tweet| %>
  <h1><%= tweet.title %></h1>
  <p><%= tweet.content %></p>
<% end %>
```

Data Base:　rake db:create
db/development.sqlite3

rails g model user name:string username:string location:string about:text

app/models/user.rb　　モデルファイル
db/migrate/20130630062417_create_users.rb　　マイグレーションファイル

アプリケーション作成の流れ

1.	作業ディレクトリの作成	$ mkdir rails
2.		$ cd rails
3.	アプリケーション作成	$ rails new demo
4.	ディレクトリ移動	$ cd demo
5.	必要な gem のインストール	$ bundle install
6.	データベースの設定	$ vi config/database.yml
7.	データベースの作成	$ rake db:create
8.	ジェネレータ	$ rails generate controller Say Hello goodbye
9.	テーブル作成	$ rake db:migrate
10.	サーバ起動	$ rails server

Action View について（ http://railsguides.jp/action_view_overview.html ）

Rails では、Web リクエストは Action Pack で取り扱われます。URL を actions に mapping する routing 機能と action を実装する controller と描画して resonse する view からなる。**Action Controller と Action View は、Action Pack を構成する 2 大要素です。この動作はコントローラ寄りの部分（ロジックの実行）とビュー寄りの部分(テンプレートの描画）に分かれます。**Action Controller は、データベースとのやりとりや、必要に応じた CRUD (Create/Read/Update/Delete) アクションの実行にかかわります。Action View はその後レスポンスを実際の Web ページにまとめる役割を担います。

Action Pack – From request to response

Action Pack is a framework for handling and responding to web requests. It provides mechanisms for **routing** (mapping request URLs to actions), defining **controllers** that implement actions, and generating responses by rendering **views**, which are templates of various formats. In short, Action Pack provides the view and controller layers in the MVC paradigm.

It consists of several modules:

- Action Dispatch, which parses information about the web request, handles routing as defined by the user, and does advanced processing related to HTTP such as MIME-type negotiation, decoding parameters in POST, PATCH, or PUT bodies, handling HTTP caching logic, cookies and sessions.
- Action Controller, which provides a base controller class that can be subclassed to implement filters and actions to handle requests. The result of an action is typically content generated from views.

With the Ruby on Rails framework, users only directly interface with the Action Controller module. Necessary Action Dispatch functionality is activated by default and Action View rendering is implicitly triggered by Action Controller. However, these modules are designed to function on their own and can be used outside of Rails.

■フォームによるパラメータの受け渡し （ http://qiita.com/To_BB/items/fe9cada1a0bcfe5e3efb ）

<% ... %>　Ruby スクリプト片をその場で実行
<%= ... %>　式を評価した結果をその場に挿入
@変数名　ビューからコントローラのインスタンス変数(@変数名)にアクセスできる

View 側

view/users/create.html.erb

```
<% form_for @user do |f| %>
 名前：<%= f.text_field :name %>
 年齢：<%= f.text_field :age %>
 職業：<%= f.text_field :job %>
<% end %>
```

form_for: 任意の model に基づいた form を作るときに使う。form_tag: model に基づかない form を作るときに使う。

このとき、あるユーザー(太郎)が

名前:太郎

年齢:24 歳

職業:学生

<form name="user">
名前：<input type="text"　name="name">
年齢：<input type="text"　name="age">
職業：<input type="text"　name="job">
</form>

と入力したとすると，この場合渡される値は以下のように格納されています。

```
params[:user][:name]        #=> 太郎
params[:user][:age]         #=> 年齢
params[:user][:job]         #=> 学生
```

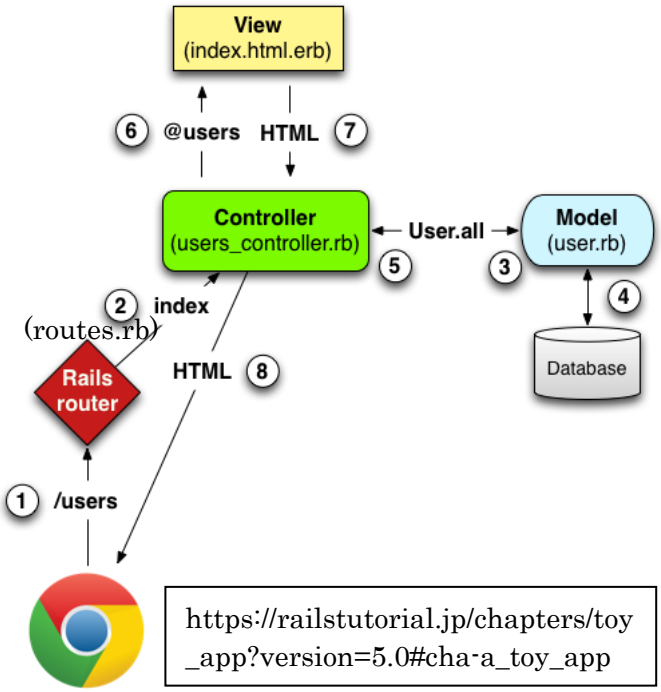
model

users	
id	integer
name	string
email	string

```
$ rails generate scaffold User name:string email:string

$ rails db:migrate
```

1. ブラウザから「/users」という URL のリクエストを Rails サーバーに送信する。
2. 「/users」リクエストは、Rails のルーティング機構（ルーター）によって **users コントローラ**内の **index** アクションに割り当てられる。
3. index アクションが実行され、そこから User モデルに、「すべてのユーザーを取り出せ」(User.all) と問い合わせる。
4. User モデルは問い合わせを受け、すべてのユーザーをデータベースから取り出す。
5. データベースから取り出したユーザーの一覧を User モデルからコントローラに返す。
6. Users コントローラは、ユーザーの一覧を@users 変数（@は Ruby のインスタンス変数を表す）に保存し、index ビューに渡す。
7. index ビューが起動し、ERB (Embedded RuBy: ビューの HTML に埋め込まれている Ruby コード) を実行して HTML を生成（レンダリング）する。
8. コントローラは、ビューで生成された HTML を受け取り、ブラウザに返す。



User リソース

model

```
$ rails generate scaffold User name:string email:string
$ rails db:migrate
```

users	
id	integer
name	string
email	string

HTTP	リクエスト	URL	アクション	用途
GET	/users	index		すべてのユーザーを一覧するページ
GET	/users/1	show		id=1 のユーザーを表示するページ
GET	/users/new	new		新規ユーザーを作成するページ
POST	/users	create		ユーザーを作成するアクション
GET	/users/1/edit	edit		id=1 のユーザーを編集するページ
PATCH	/users/1	update		id=1 のユーザーを更新するアクション
DELETE	/users/1	destroy		id=1 のユーザーを削除するアクション

表 2.2: リスト 2.5 の Users リソースが提供する RESTful なルート

②*app/controllers/users_controller.rb*

```
class UsersController < ApplicationController
  .
  def index
    @users = User.all
  end
  def show
  end
  def new
  end
  def edit
  end
  def create
  end
  def update
  end
  def destroy
  end
end
```

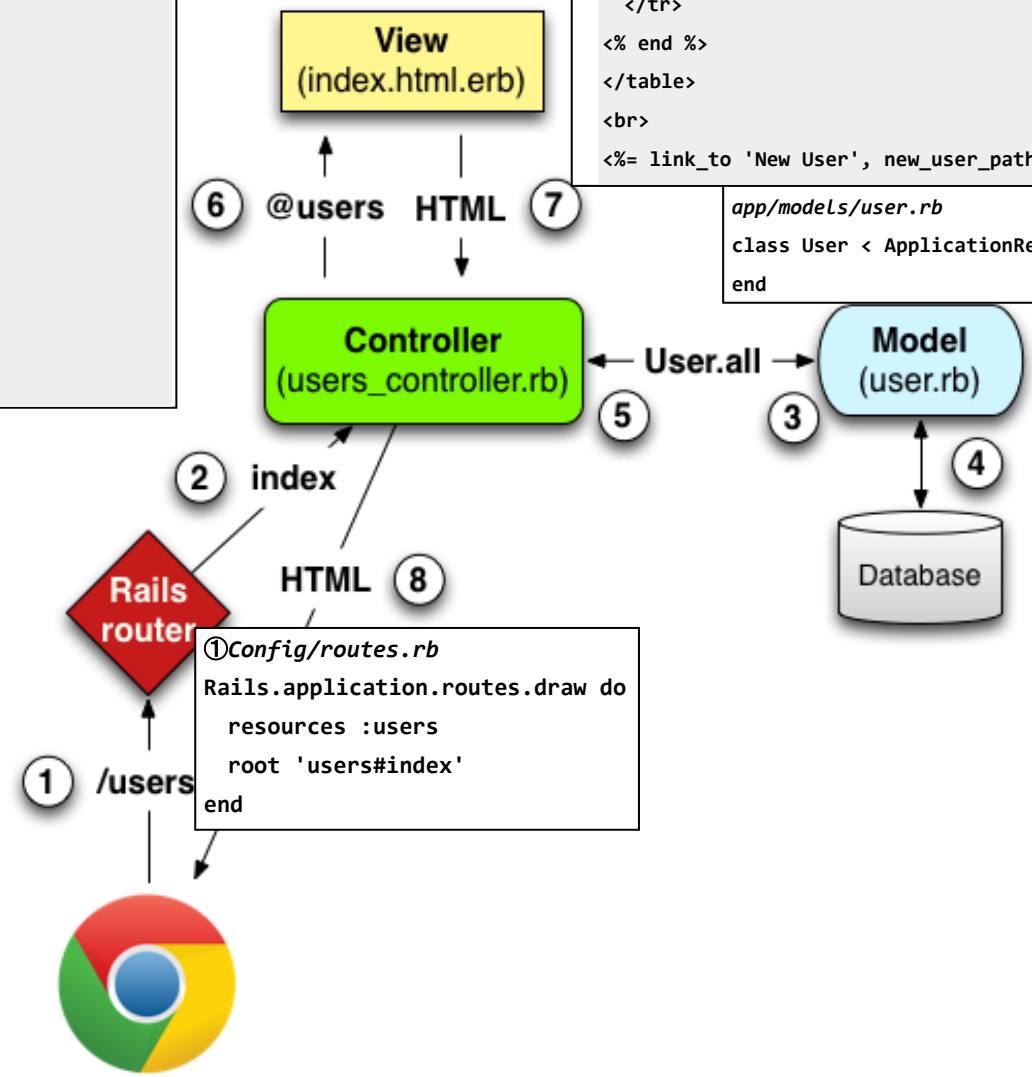
⑦*app/views/users/index.html.erb*

```
<h1>Listing users</h1>
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email</th>
      <th colspan="3"></th>
    </tr>
  </thead>
```

```
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit',
edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user,
            method: :delete,
     data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>
<br>
<%= link_to 'New User', new_user_path %>
```

app/models/user.rb

```
class User < ApplicationRecord
end
```



Microposts リソース

model

users	
id	integer
name	string
email	string

```
$ rails generate scaffold Micropost content:text user_id:integer

$ rails db:migrate
```

HTTP	リクエスト	URL	アクション	用途
GET	/microposts	index		すべてのマイクロポストを表示するページ
GET	/microposts/1	show		id=1 のマイクロポストを表示するページ
GET	/microposts/new	new		マイクロポストを新規作成するページ
POST	/microposts	create		マイクロポストを新規作成するアクション
GET	/microposts/1/edit	edit		id=1 のマイクロポストを編集するページ
PATCH	/microposts/1	update		id=1 のマイクロポストを更新するアクション
DELETE	/microposts/1	destroy		id1 のマイクロポストを削除する

表 2.3: Microposts リソースが提供するリスト 2.11 の RESTful ルート

②app/controllers/microposts_controller.rb

```
class MicropostsController < ApplicationController

  def index

  end
  def show
  end
  def new
  end
  def edit
  end
  def create
  end
  def update
  end
  def destroy
  end
end
```

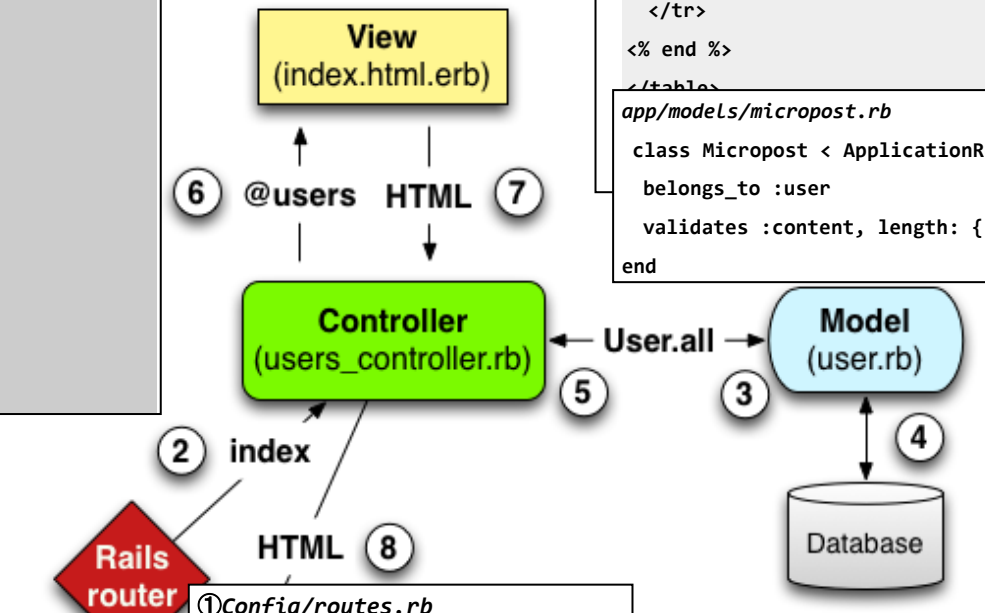
⑦app/views/users/index.html.erb

```
<h1>Listing users</h1>
<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Email</th>
      <th colspan="3"></th>
    </tr>
  </thead>
```

```
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
    <td><%= link_to 'Show', user %></td>
    <td><%= link_to 'Edit',
edit_user_path(user) %></td>
    <td><%= link_to 'Destroy', user,
            method: :delete,
     data: { confirm: 'Are you sure?' } %></td>
  </tr>
<% end %>
</table>
```

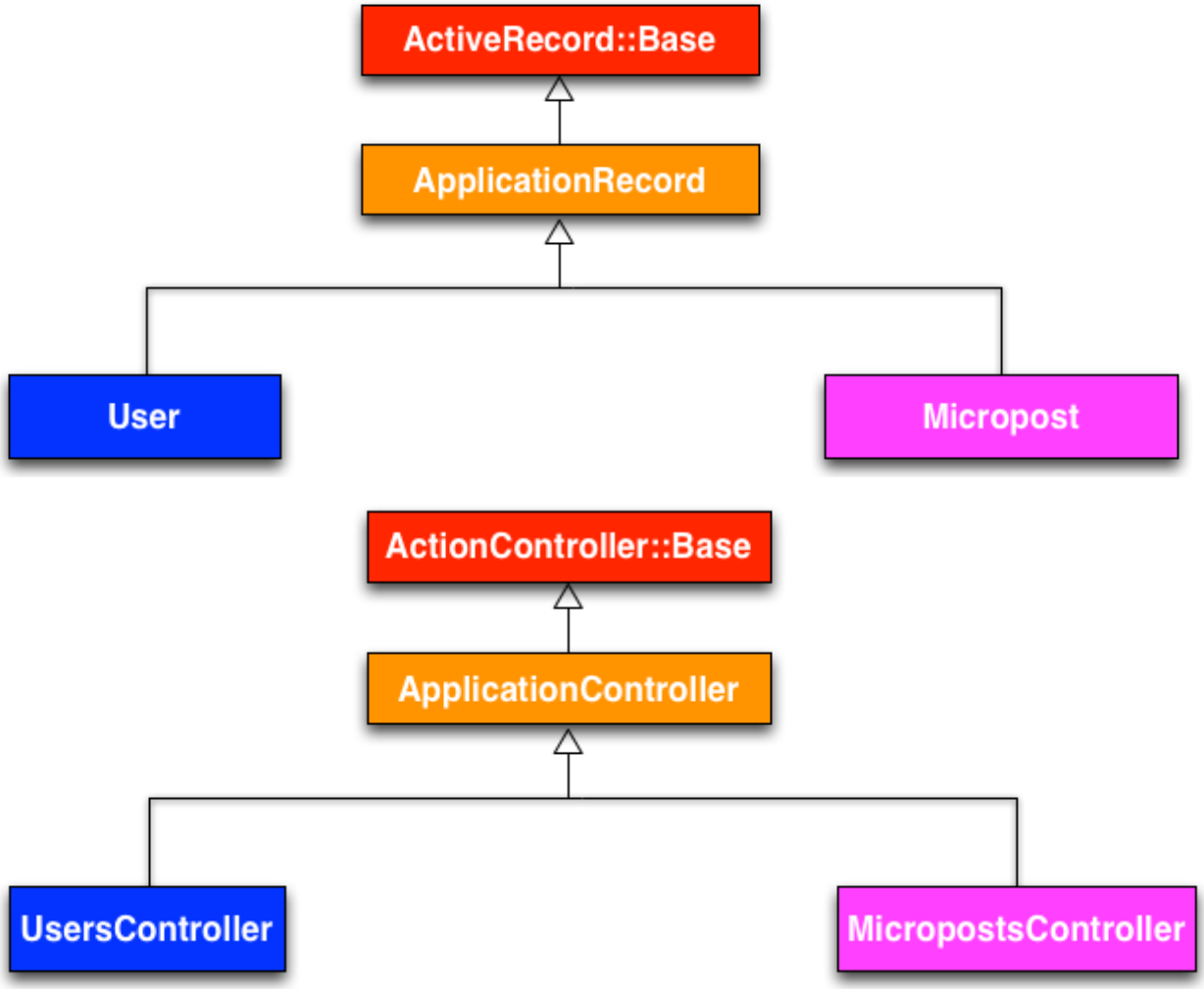
app/models/micropost.rb

```
class Micropost < ApplicationRecord
 belongs_to :user
 validates :content, length: { maximum: 140 }
end
```



①Config/routes.rb

```
Rails.application.routes.draw do
  resources :microposts
  resources :users
  root 'users#index'
end
```



StaticPagesController は Ruby のクラスですが、ApplicationController クラスを継承しているため、StaticPagesController のメソッドは (たとえ何も書かれていなくても) Rails 特有の振る舞いをします。具体的には、/static_pages/home という URL にアクセスすると、Rails は StaticPages コントローラを参照し、home アクションに記述されているコードを実行します。その後、そのアクションに対応するビュー (1.3.3 で説明した MVC の V に相当) を出力します。今回の場合、home アクションが空になっているので、/static_pages/home にアクセスしても、単に対応するビューが出力されるだけです。

ActiveSupport

クラス名とファイル名の対応

クラス名をアンダースコア化したファイル名にそのクラスを定義する

→オートロード

ActiveRecord

Rails で params を使ってデータを取得する

目次

params とは

■概要
Rails では、リクエスト情報をひとまとめにして、params[:パラメータ名]という形式で取得できます。
[リクエスト情報とは、実際どういったものか？]
代表的なものは以下の2つ
・POST でフォームから送信されたデータ
・クエリ情報

■特徴
・ただの文字列だけではなく、配列やハッシュを受け取ることもできる
・リンクによるパラメータの受け渡しができる
・フォームによるパラメータの受け渡しができる

具体例

■リンクによるパラメータの受け渡し
View 側

view/users/show.html.erb
```
<%= link_to 'ユーザ名', controller: 'users', action: 'show',
           id: 1, name: '太郎' %>
```

Controller 側

controller/users_contoller.rb
```
def show
 id = params[:id] #=> id = 1
 name = params[:name] #=> name = '太郎'
end
```

以上がリンクによるパラメータの受け渡しの基本的な例です。

■フォームによるパラメータの受け渡し
View 側

view/users/create.html.erb

```
<% form_for @user do |f| %>
 名前：<%= f.text_field :name %>
 年齢：<%= f.text_field :age %>
 職業：<%= f.text_field :job %>
<% end %>
```

このとき、あるユーザー(太郎)が
名前：太郎
年齢：24 歳
職業：学生
と入力しました。すると
この場合渡される値は以下のように格納されています。

```
params[:user][:name] #=> 太郎
params[:user][:age] #=> 年齢
params[:user][:job] #=> 学生
```

■クエリ情報をパラメータで渡す場合
//localhost:3000/users/show?id=1&age=24
上記のような URL リクエストの場合、渡される値は以下です。

```
params[:id] #=> 1
params[:age] #=> 24
```

クエリ情報がハッシュのようなカタチで格納されています。

配列とハッシュについて

■配列
配列を渡す場合は、キー名の末尾に[]を付与する必要があります。
(例)
~~/?group[]=ruby&group[]=php&group[]=java
この時、実際どういう具合に値が渡されているか

```
params[:group].inspect #inspect メソッドは取得した配列を整形してくれる
=> ["ruby", "php", "java"]  #配列 group が格納されている。


# "ruby"だけを取り出したい場合
params[:group][0] #=> "ruby"
```

■ハッシュ
先ほどのまでの話と重複しているが、復習程度に。

```
params = ActionController::Parameters.new(name: '太郎',
                           email: 'taro@example.com')

# name&email をそれぞれキーとして
params[:name] #=> '太郎'
params[:email] #=> taro@example.com
```

以上が基本的な params の概念、使い方です。

参考文献

Ruby on Rails 4 アプリケーションプログラミング，山田 祥寛
http://www.amazon.co.jp/dp/4774164100
Rails3 レシピブック 190 の技，高橋 征義
http://www.amazon.co.jp/dp/4797363827

Rails リファレンス
http://railsdoc.com/references

Rails
ActiveSupport
クラス名とファイル名の対応
Members::ToDoList　　members/to_do_list.rb

Ruby

コメント
以降行末までコメント
=begin
から
=end
までコメント

module Foo
　　class Bar
　　end
end
Foo::Bar

String
'Hellow' + 'Ruby'
"Ruby since #{2016 - 1995}"　　""の中の#{}の内部は Ruby コードとして処理される。
str = 'test'
str.delete('t')　　str→'test'
str.detelte!('t')　　str→'es'

Symbol 変更できない文字情報を表す

:name

IO
File
Array
array = ['b', 'a', 'c']
array.size
array.sort　　['a', 'b', 'c']
array.each{ |i| puts i}
[1,2,3].map{ |i| i*10 }　　[10, 20, 30]
words =　['I', 'like, 'Rails']

Hash
{:hobby => 'programming'} puts I[:hobby]

定数　ADULT_AGE = 18
グローバル変数　$ で始まる　$var
ローカル変数　小文字で始まる　name = '近大'　　puts name
クラス変数　@@で始まる　@@var
インスタンス変数　　@で始まる　@var

```
def   name(parameter)
        # method body
end

if   input == password

else

end
```

Integer#upto　　Integer クラスのインスタンスメソッド upto
Fixnum.name　　Fixnum クラスのクラスメソッド name　　（Fixnum::name と書くこともある）

```
1.upto(100) do |i|          # |i| のiブロック引数
        puts i
end
```