プロトレ応用1　〜タスク1. アプリ新規作成手順の確認〜

Lesson7「FirstApp」の作成手順書

　●新規アプリケーションを作成する

　●一覧機能（indexアクション）を実装する

　●新規投稿画面（newアクション）を実装する

　●保存機能（createアクション）を実装する

～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～

アプリケーションの開発プロセスは、プロジェクトや使用するフレームワークによって若干異なることがありますが、一般的なMVC（モデル・ビュー・コントローラー）アーキテクチャを用いる場合、以下の手順をたどるのが効率的です。

**1. モデルの作成**

まず、データの構造（テーブルのカラムやデータ型）について設計します。モデルクラスを作成して、それがどのテーブルに対応するか、どのカラムを持つかを定義します。

**2. テーブルの作成**

次に、データベースに実際のテーブルを作成します。SQLのクエリやフレームワーク特有のマイグレーションスクリプトを使ってテーブルを作成し、カラムや制約を設定します。

**3. ルーティングの設定**

アプリケーションのURLパターンを定義し、どのコントローラーとアクションに紐付けるかを設定します。これにより、どのURLがどの機能に対応するかが決まります。

**4. コントローラーの作成**

ルーティングに従って、各URLに対応するコントローラーを作成します。必要に応じてファイルを作成し、基本的なアクション（例：index、show、create、update、destroy）を定義します。

**5. コントローラーの設定**

具体的なロジックをコントローラーの各アクションに記述します。モデルと連携しつつ、データの取得や保存、削除などの操作を実装します。

**6. ビューの作成**

コントローラーのアクションに対応するビューを作成します。例えば、HTMLテンプレートを作成して、データを表示するための準備をします。

**7. ビューの設定**

作成したテンプレートに動的にデータを挿入するロジックを記述します。必要に応じて、ループや条件分岐を用いてデータを表示します。

一例を含む具体的な手順

**モデルの作成:**

例えば、ブログ記事のアプリケーションを作る場合、Article（記事） というモデルクラスを定義します。

**テーブルの作成:**

articles というテーブルをデータベースに作成し、カラム（title、body、created\_atなど）を設定します。

**ルーティングの設定:**

GET /articles で記事の一覧を表示するルート、POST /articles で新しい記事を作成するルートなどを設定します。

**コントローラーの作成:**

ArticlesController を作成し、index、show、new、create などのアクションを定義します。

**コントローラーの設定:**

index アクションで全記事を取得し、ビューに渡すロジックを記述します。create アクションで新しい記事をデータベースに保存するロジックを記述します。

**ビューの作成:**

index.html.erb などのテンプレートを作成し、記事の一覧を表示するコードを書きます。

**ビューの設定:**

@articles という変数を使って、データベースから取得した記事データを動的に表示します。

～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～～

# 1. アプリケーションの新規作成手順

①Railsアプリケーションの新規作成

②データベースの作成

## ①Railsアプリケーションの新規作成

1, ターミナルで新規アプリを作成するディレクトリに移動

**【cd ~/projects】を実行**

2, projectsディレクトリ内に、FirstAppを新規作成

**【rails \_7.0.0\_ new first\_app -d mysql】を実行**

・rails newアクションで新規アプリを作成する。

・アプリケーション名の後にオプションを付けることが出来る。

※オプション…コマンドの動作の変更、追加ができる。

**★rails new アプリケーション名 （– オプション名）**

・アプリケーションがきちんと作成できているか、確認のためディレクトリを表示。

　cd first\_app　→　pwd

　/Users/ユーザー名/projects/first\_app　と表示されていたら作成完了。

## ②データベースの作成

1, 新規データベースを作成

**【rails db:create】を実行**

・データベースを可視化するためには、Sequel Pro（Mac）/DBeaver（Windows）を活用する。

※データベースの削除

**★rails db:drop**

2, ローカルサーバーの起動

**【rails s】を実行**

・サーバーを停止するためには「Ctrl＋C」を行う。

# 2. 一覧機能（indexアクション）を実装する手順

①ルーティングの設定

②コントローラーの作成および設定

③ビューの作成

④モデルの作成

## ①ルーティングの設定

1, config/routes.rb　を編集

【Rails.application.routes.draw do

**get 'posts', to: 'posts#index'**

end】

**★HTTPメソッド　'URIパターン',　to:　'コントローラー名#アクション名'**

※HTTPメソッド…GET（取得）

POST（送信、作成）

PUT（更新、作成）

DELETE（削除）

※コントローラー名…複数形を使用（今回は「投稿する」という意味を持つ「posts」を使用）

※アクション…index（一覧表示ページの表示）

new（新規投稿ページの表示）

create（データの投稿を行う）

show（個別詳細ページの表示）

edit（投稿編集ページの表示）

update（データの編集を行う）

destroy（データの削除を行う）

2, ルーティングの確認

**【rails routes】を実行**

Prefix Verb URI Pattern Controller#Action

posts GET /posts(.:format) posts#index　　　　と表示されていれば設定完了。

## ②コントローラーの作成および設定

1, コントローラーの作成

**【rails g controller posts】を実行**

**★rails g controller(ファイルの種類) 生成するファイル名**

※ファイルの削除

**★rails d ファイルの種類 削除するファイル名**

・ｇ…generate「生成する」の略

　　↓

　app/controllers/posts\_controller.rb　が生成されていれば完了。

2, コントローラーの設定（アクションの定義）

・app/controllers/posts\_controller.rb　を編集

【class PostsController < ApplicationController

**def index**

end】

## ③ビューの作成

1, ビューファイルの作成

・app/views/postsというディレクトリ内に**【index.html.erb】**というファイルを作成。

※ERB…テンプレートエンジンと呼ばれる仕組みを持ったものの1つ。

　　　　　雛形となるテンプレートと、そのテンプレートにデータとなる記述を埋め込むことで、

　　　　　最終的に別のファイルとしてデータを含んだHTMLファイルを生成することができる。

**★ (アクション名).html.erb**　という名前で作成

2, app/views/posts/index.html.erb　を編集

**【<h1>トップページ</h1>】**

3, コントローラーでインスタンス変数を定義

app/controllers/posts\_controller.rb

【class PostsController < ApplicationController

def index

**@post = "これはコントローラーで定義したインスタンス変数を確認するための文字列です"**

end

end】

※コントローラーのアクション内にインスタンス変数を定義しておくことで、

　　その情報を持つ変数をビューファイル内でも使用することが出来る。

4, ビューファイルを編集

app/views/posts/index.html.erb

【<h1>トップページ</h1>

**<%= @post %>** 】

※ブラウザ（[**http://localhost:3000/posts**](http://localhost:3000/posts)）を確認すると、以下のような表示がされている。



**★<%= %>** …画面上に表示させたいものの入力

**<% %>** …画面上に表示されなくても良いもの（if文やeach分など）

## ④モデルの作成

1, モデルの作成　（→5, に飛んでも良い）

**【rails g model post】を実行**

**★rails g model モデル名**

※モデル名…大文字で始まる単数形を用いる。

（→データベース上の名前は小文字の複数形になる）

　　↓

app/models/post.rb　が生成される。（同時に複数のファイルも生成されている。）

2, テーブルの仕様を決める（マイグレーションファイルの編集）

・モデルを作成した際に「**db/migrate/(日付)\_create\_posts.rb**」というファイルも生成されている。

　この「db/migrate/」ディレクトリに入っているファイルは、**マイグレーション**という役割を持っている。

※マイグレーション…テーブルの設計図・仕様書

　　　　　　　　　　　マイグレーションファイルの情報が反映されて、テーブルの作成・変更ができる。

※テーブル…データベース内に作成される、表形式のデータの収納場所。

　　　　　　　（データベースには直接データを保存できないため）

　　　　　　　テーブルはモデルに対応している＝テーブルの分だけモデルが存在する。

レコード（行）、カラム（列）で表現する。

・マイグレーションファイルを編集

db/migrate/20XXXXXXXXXXXX\_create\_posts.rb

【class CreatePosts < ActiveRecord::Migration[6.0]

def change

create\_table :posts do |t|

**t.text :memo**

t.timestamps

end

end

end】

※「create\_table :posts do |t|」…「実際にpostsテーブルを作成する」という仕様が記載されている。

※カラムの型…「**t.**」の後にカラムの型、カラム名（シンボルで記載）を続ける。

　　　　　　　　integer（数値：金額、回数など）

string（文字(短文)：ユーザー名、メールアドレスなど）

text（文字(長文)：投稿文、説明文など）

boolean（真か偽か：はい/いいえの選択、合格/不合格のフラグなど）

datetime（日付と時刻：作成日時、更新日時など）

3, テーブルの作成（マイグレーションの実行）

**【rails db:migrate】を実行**

4, テーブルの確認

Sequel Pro（Mac）やDBeaver（Windows）で「posts」テーブルが作成されているかを確認。

「posts」テーブルをクリックすると、テーブルの中身を確認できるので、

その中に「memo」カラムが追加されていることも確認。

※マイグレーションの修正…マイグレーションは、一度実行すると再度実行できない仕組みになっているため、

**★rails db:rollback**　　　　修正を行いたい場合は、差し戻す必要がある。

※マイグレーションの状況確認…マイグレーションが実行済みかどうかを確認する。

**★rails db:migrate:status**　　 Statusがupになっていれば実行済み（downにであればロールバックせずに修正可）

5, マイグレーションファイルの修正、実行、確認

・**【rails db:rollback】を実行して差し戻す。**

・db/migrate/(日付)\_create\_posts.rb　を編集

【class CreatePosts < ActiveRecord::Migration[7.0]

def change

create\_table :posts do |t|

**t.text :content**※:memo → :content に変更

t.timestamps

end

end

end】

・**【rails db:migrate】を実行**

・テーブルを確認

「memo」カラムが「content」カラムに変更されている。

※＋α　コンソール（ターミナルから直接アプリケーションを操作する方法）

**★rails c** …コンソールを起動

※ActiveRecordメソッド…モデルがテーブル操作に関して使用できるメソッドの総称。

　　　　　　　　　　　　all（テーブルすべてのデータを取得）

find（引数にレコードのidを指定し、対応するレコードを取得）

new（クラスのインスタンス(レコード)を生成する）

save（クラスのインスタンス(レコード)を保存する）

　　irb(main):001:0> post = Post.new　※「Post」はモデル名

irb(main):002:0> post.content = "こんにちは！"　※インスタンスにデータを追加

irb(main):003:0> post.save　※そのモデルが対応するテーブルにレコードを保存

irb(main):007:0> exit　※コンソール終了

―――

irb(main):001:0> Post.all　※全レコードの取得

irb(main):002:0> Post.find(1)　※単一レコードの取得

irb(main):003:0> post = Post.find(1)　※レコードを変数に代入

irb(main):004:0> post　※変数でレコードを表示

=> #<Post id: 1, content: "はじめまして", created\_at: "2019-01-01 00:00:00", updated\_at: "2019-01-02 00:00:00">

irb(main):005:0> post.content　※カラムデータの表示

=> "はじめまして"

6, コントローラーで複数レコードを取得

・app/controllers/posts\_controller.rb　を編集

【class PostsController < ApplicationController

def index

**@posts = Post.all** ※すべてのレコードを@posts（複数レコードを取得するため複数形）に代入

end

end】

7, eachメソッドで複数レコードのカラムデータを1つずつ表示する

・app/views/posts/index.html.erb　を編集

【<h1>トップページ</h1>

**<% @posts.each do |post| %>**

**<%= post.content %>**　※ブロック変数（|post|）内にある「post」を使用

**<%= post.created\_at %>**

<% end %>】

8, 見た目を整える

・app/views/posts/index.html.erb　を編集

【<h1>トップページ</h1>

<% @posts.each do |post| %>

**<div class="post">**

**<div class="post-date">**

**投稿日時：<%= post.created\_at %>**

**</div>**

**<div class="post-content">**

**<%= post.content %>**

**</div>**

**</div>**

<% end %>】

・CSSファイルを作成

app/assets/stylesheetsというディレクトリ内に、**【posts.css】**というCSSファイルを作成。

・app/assets/stylesheets/posts.css　を編集

**.post{**

**border: 1px solid;**

**width: 40%;**

**margin-top: 30px;**

**}**

**.post-date{**

**color: gray;**

**}**

※ブラウザ（[**http://localhost:3000/posts**](http://localhost:3000/posts)）を確認すると、以下のような表示がされている。



# 3. 投稿画面（newアクション）を作成する手順

①ルーティングの設定

②コントローラーの設定

③ビューの作成と設定

## ①ルーティングの設定

・config/routes.rb　を編集

【Rails.application.routes.draw do

get 'posts', to: 'posts#index'

**get 'posts/new', to: 'posts#new'**

end】

・**【rails routes】を実行**

　ルーティングが正しく設定できていることを確認。

## ②コントローラーの設定

・app/controllers/posts\_controller.rb　にアクションを定義

【class PostsController < ApplicationController

def index

@posts = Post.all

end

**def new**

**end**

end】

## ③ビューの作成と設定

1, 新規投稿ページを作成

・app/views/postsというディレクトリに、**【new.html.erb】**を作成。

・app/views/posts/new.html.erb　を編集

**【<h1>新規投稿ページ</h1>**

**<%= form\_with url: "/posts", method: :post, local: true do |form| %>**

**<%= form.text\_field :content %>**　※ブロック変数（|form|）内にある「form」を使用

**<%= form.submit '投稿する' %>**

**<% end %>】**

※ヘルパーメソッド…Railsにおける、主にビューでHTMLタグを出現させたりテキスト加工するために使用するメソッドの総称。

　　　　　　　　　　form\_with（投稿ページなどにおけるフォームの実装）

link\_to（リンクの実装）

※form\_withに指定する引数について

　　　　　　　　　　urlオプション（フォームの情報を送るリクエストのパスを指定）

methodオプション（フォームの情報を送るリクエストのHTTPメソッドを指定。

オプションの初期値は「:post」のため「postメソッド」を指定する場合は省略が可能。）

local（リモート送信を無効にするかどうかを指定。「true」にすると無効。）

※フォームの部品を生成する主なヘルパーメソッド

　　　　　　　　　　text\_field（１行のテキストボックス）

password\_field（パスワード入力ボックス）

check\_box（チェックボックス(複数選択可)）

radio\_button（ラジオボタン(複数の中から１つだけ選択)）

submit（送信ボタン）

※ブラウザ（[**http://localhost:3000/posts/new**](http://localhost:3000/posts/new)）を確認すると、以下のような表示がされている。



2, 新規投稿ページへのリンクを作成

・**【rails routes】を実行**して、ルーティングを確認

Prefix Verb URI Pattern Controller#Action

posts\_new GET /posts/new(.:format) posts#new

・app/views/posts/index.html.erb　を編集

【<h1>トップページ</h1>

**<%= link\_to '新規投稿', '/posts/new' %>**

<% @posts.each do |post| %>

<div class="post">

<div class="post-date">

投稿日時：<%= post.created\_at %>

</div>

<div class="post-content">

<%= post.content %>

</div>

</div>

<% end %>】

※link\_toメソッド…HTMLのaタグの代わりに使用できる。

**★<%= link\_to 'リンクに表示する文字', 'リンク先のURL' %>**　※リンク先をURLで指定する場合

**★<%= link\_to 'リンクに表示する文字', 'パス', method: :HTTPメソッド %>**　※リンク先をパスで指定する場合

※本来であれば、

**<%= link\_to '新規投稿', '/posts/new', method: :GET %>**

　だが、link\_toはデフォルトでGETメソッドが設定されているため、省略が可能。

# 4. 保存機能（createアクション）を実装する手順

①ルーティングの設定

②コントローラーの設定

③ビューの作成

## ①ルーティングの設定

1, config/routes.rb　を編集

【Rails.application.routes.draw do

get 'posts', to: 'posts#index'

get 'posts/new', to: 'posts#new'

**post 'posts', to: 'posts#create'**※HTTPメソッド「POST」（データ投稿されたものをテーブルに登録する）

end】　　　　　　　　　　　　　アクション名「create」（投稿内容を保存してレコード作成の処理を行う）

2, ルーティングの確認

**【rails routes】を実行**

Prefix Verb URI Pattern Controller#Action

(posts\_new) POST /posts(.:format) posts#create　　　　と表示されていれば設定完了。

## ②コントローラーの設定

・app/controllers/posts\_controller.rb　にアクションを定義

【class PostsController < ApplicationController

def index

@posts = Post.all

end

def new

end

**def create**

**Post.create(content: params[:content])**

**redirect\_to "/posts"**

**end**

end】

※パラメーター…リクエストに含まれてサーバーの外部から渡されるデータのこと。

　　params…送られたパラメーターをハッシュのような構造で格納したもの

　　　　　　text\_fieldの後に記載されたキーでparamsに格納されていて、

　　　　　　フォームで入力された情報の値は、params[:キー名]として取り出すことができる。

※createメソッド…モデルが使用できるActiveRecordメソッドのひとつ。

**★モデル.create(カラム名: 値(paramsとして送られてきたデータ)**

## ③ビューの作成

※redirect\_toメソッド…あるアクションの実行後に、特定のURLへ移動するために使用