

On the Rails.

railsを使ってみよう

今回のゴール

- Rails プロジェクトのディレクトリ構成を説明できる
- `rails c` で Model を触って挙動を確認できる
- モデル・ビュー・コントローラ（MVC）の役割を説明できる
- ルーティング → コントローラ → ビューの流れを説明できる
- Controller で外部 API を呼び、結果を View に渡せる
- 部分テンプレートを使って View を分割できる

Rails とは

- Ruby で書かれた Web アプリケーションフレームワーク
- Web アプリ開発に必要な機能が一式そろっている
- 「設定より規約 (Convention over Configuration)」という考え方

Ruby 研修で学んだ文法・クラス・メソッドはそのまま Rails でも使われます

アプリケーションを作ってみよう

■ dockerを使える様にする

```
$ brew install colima  
$ colima start
```

■ 作業ディレクトリの作成

```
$ mkdir rails-training-app  
$ cd rails-training-app
```

ディレクトリ名を `rails-training-app` にする場合

■ Containerの作成＆起動

Dockerfile.dev を rails-training-app 内に設置した後に以下を実行

以下のコマンドは rails-training-app ディレクトリ内で行う

```
docker build -f Dockerfile.dev -t rails-training .
docker run -it --rm -v $(pwd):/app -p 3000:3000 rails-training bash
```

Rails アプリケーションの作成

```
# カレントディレクトリにRails作成 (SQLite)
/app# rails new . --force --database=sqlite3 --skip-test
```

- root 直下に Gemfile を作成する

```
/app# bundle install
/app# rails server -b 0.0.0.0
```

- http://localhost:3000 にアクセス
- Rails の初期画面が表示されれば成功

Rails プロジェクトの構成

```
root/
|- app/                      # アプリケーションの本体
|   |- controllers/        # リクエストを処理
|   |- models/              # データ・ロジック
|   |- views/                # 画面 (HTML + Ruby)
|- config/                   # ルーティングや各種設定
|- db/                       # データベース関連
```

app 配下にはこれ以外にも assets や helpers があります。
任意のディレクトリを作成することもできます。

https://railsguides.jp/getting_started.html#ディレクトリ構造

■ `./config/` の中身を見てみよう

```
/app# ls -la config
```

```
config/
|- environments/      # アプリケーションの本体
|- initializers/     # アプリケーション実行時に一度ロードされる
|- locales/          # 言語毎の表示文言管理
|
|- application.rb    # アプリケーションの設定
|- database.yml       # データベースへの接続設定
|- routes.rb          # ルーティングの制御
```

■ `rails console` でコードを実行する

```
/app# rails c
```

- Rails アプリの中身を対話的に実行できる
- irb の Rails 版

■ 現在時刻を確認してみよう

```
app(dev):001> Time.current
```

scaffold でリソース生成

記事データを扱えるように関連ファイルを作成してみよう

`rails console` を `exit` で抜け、以下を実行する

```
/app# rails g scaffold Article title:string body:text  
/app# rails db:migrate
```

マイグレーションファイルやModel、Viewなどの必要ファイルを自動生成してくれる

できたら <http://localhost:3000/articles> にアクセスしてみよう

MVCを理解する

- **Model**：データやビジネスロジックを扱う（Ruby クラス）
- **Controller**：リクエストを受けて処理を振り分ける
- **View**：画面表示（HTML + Ruby）

簡単にいうと「役割分担」

ルーティングを確認

```
/app# rails routes
```

```
# config/routes.rb
get "articles", to: "articles#index"
```

- URL と Controller#action を結びつける
- 「どのリクエストを、どの処理に渡すか」を定義

API って何？

- 外部サービスとデータをやり取りする仕組み
- JSON 形式でデータを返すことが多い

今回は「外部からデータを取得して表示する」体験をします

rails console で呼び出してみよう

```
require "net/http"
require "json"

# Zenn ユーザ情報 API (ユーザー名を指定)
username = "zenn"
url = URI("https://zenn.dev/api/users/#{username}")
response = Net::HTTP.get(url)
user_info = JSON.parse(response)
```

- Array / Hash の復習

Controller で API を実行する

```
# app/controllers/articles_controller.rb

def index
  require "net/http"
  require "json"
  # Zenn ユーザ情報 API
  username = "zenn"
  url = URI("https://zenn.dev/api/users/#{username}")
  response = Net::HTTP.get(url)
  @user_info = JSON.parse(response)
end
```

View に表示してみよう

```
<!-- app/views/articles/index.html.erb -->

<h2>Zenn ユーザー情報</h2>
<% if @user_info.present? %>
  <ul>
    <li>ユーザー名: <%= @user_info["username"] %></li>
    <li>表示名: <%= @user_info["name"] %></li>
    <li>フォロワー数: <%= @user_info["followers_count"] %></li>
  </ul>
<% else %>
  <p>Zenn のユーザー情報を取得できませんでした。</p>
<% end %>
```

応用：記事一覧 API を使ってみよう

別ページ `articles` に記事一覧を表示してみましょう

エンドポイントは以下を使用してください

```
https://zenn.dev/api/articles?username=(ユーザー名)
```

<https://zenn.dev/api/users/zenn> のレスポンスは Hash

<https://zenn.dev/api/articles?username=zenn> のレスポンスは Array (複数記事)

表示する記事をフィルタリングしてみよう

表示する記事を **最新記事5件** と **人気記事5件** にしてみましょう
それぞれ別項目として表示してください

同一ページ内に表示してください

部分テンプレートとレイアウト

- View が大きくなると管理が大変
- Rails では View を分割して整理できる

部分テンプレートって何？

- View の一部を切り出したファイル
- 繰り返し使う UI をまとめられる

Ruby の「メソッド化」と同じ考え方 (DRY)

部分テンプレートを使ってみよう

以下の部分テンプレートを作ってみましょう

```
<!-- app/views/articles/_article.html.erb -->

<li>
  <%= article["emoji"] %>
  <%= article["title"] %>
  (❤ <%= article["liked_count"] %>)
</li>
```

部分テンプレートを使ってみよう

先ほど作成した部分テンプレートを埋め込んでみましょう

```
<!-- app/views/articles/index.html.erb -->

<ul>
  <% @popular_articles.each do |article| %>
    <%= render partial: "article", locals: { article: article } %>
  <% end %>
</ul>
```

レイアウトとは？

- ページ共通の枠組み
- ヘッダーやフッターをまとめる

```
<%= yield %>
```

- 各 View の内容が差し込まれる

レイアウトを使ってみよう

```
<!-- app/views/layouts/application.html.erb -->

<body>
  <header>Sample App</header>
  <%= yield %>
</body>
```

- 各 View は `yield` の位置に表示される

本日のおさらい

- Rails プロジェクトのディレクトリ構成を説明できる
- `rails c` で Model を触って挙動を確認できる
- モデル・ビュー・コントローラ（MVC）の役割を説明できる
- ルーティング → コントローラ → ビューの流れを説明できる
- Controller で外部 API を呼び、結果を View に渡せる
- 部分テンプレートを使って View を分割できる

Rails プロジェクトのディレクトリ構成を説明できる

- Rails アプリの中心はどのディレクトリ？
- MVC に関するファイルはどこに置かれる？
- config や db は何のために存在している？

ヒント：普段触るコードの多くがあるのは...

rails c で Model を触って挙動を確認できる

- irb と何が同じで、何が違う？
- どんな場面で使うと便利？

モデル・ビュー・コントローラ (MVC) の役割を説明できる

- データを扱うのはどれ？
- 処理の流れを制御するのはどれ？
- HTML を書くのはどれ？

ヒント：「何をするか」で責務が分かれています

ルーティング → コントローラ → ビューの流れを説明できる

- 最初に呼ばれるのはどこ？
- URL と Controller はどうやって結びつく？
- View はいつ描画される？

ヒント：リクエストは上から順に流れます

Controller で外部 API を呼び、結果を View に渡せる

- API 呼び出しへどこに書いた？
- View では何をしてはいけない？
- @posts はどこから来た？

ヒント：Controller は「データ準備係」です

部分テンプレートを使って View を分割できる

- どんな View を切り出すと良さそう？
- Ruby のどんな考え方と似ている？
- 分割すると何が嬉しい？

ヒント：同じ HTML を何度も書かないための仕組みです

On the Rails.

On the Rails.

レールに乗って、どこへ行く？