【Ruby on Rails】基礎

目次

前提知識編

■Ruby言語

▶ ※ RubyのNotionページを見よ。

■Railsに特化した知識

▶ ※ `require` を書く機会はほとんどなくなる。

▶ ※ 未定義のモジュールを自動的に作成してくれる（物理的なファイルが作成されることはない）。

▶ ※ Ruby標準クラスの拡張がふんだんに行われている。

▶ ※ Rails は非常に変化が速いフレームワークである。

▶ ※ MVC (Model/View/Contoller)、REST (Representaional State Transfer)、デザインパターンについての知識が必要。

■Web技術

▶ ※ TCP/IP、HTTP、WebSocket、SSL/HTTPS、HTML、JavaScript、Ajax、CSS についての知識が必要。

▶ ※ HTMLはその拡張言語である ERB や HAML などで記述することが多い。

▶ ※ CSSはその拡張言語である Sass や LESS などで記述することが多い。

▶ ※ JavaScriptはその拡張言語である CoffeeScript などで記述することが多い。

■データベース技術

▶ ※ RailsがRDBMSの機能をうまくラップしてくれているため、Railsアプリ開発中にSQLを書いたり、RDBMS固有の機能を直接操作したりする場面は非常に少ない。ただし、Railsが提供しているデータベース関連の機能を理解しておく必要がある。

■セキュリティ

▶ ※ Railsアプリケーションに関するセキュリティ上の考慮点について、このドキュメントを参照して必ず理解せよ。

【Ruby on Rails】基礎

目次

- 前提知識編
- 環境構築編
- 基本編

前提知識編

■Ruby言語

▶ ※ RubyのNotionページを見よ。

■Railsに特化した知識

▶ ※ `require` を書く機会はほとんどなくなる。

▶ ※ 未定義のモジュールを自動的に作成してくれる（物理的なファイルが作成されることはない）。

▶ ※ Ruby標準クラスの拡張がふんだんに行われている。

▶ ※ Rails は非常に変化が速いフレームワークである。

▶ ※ MVC (Model/View/Contoller)、REST (Representaional State Transfer)、デザインパターンについての知識が必要。

■Web技術

▶ ※ TCP/IP、HTTP、WebSocket、SSL/HTTPS、HTML、JavaScript、Ajax、CSS についての知識が必要。

▶ ※ HTMLはその拡張言語である ERB や HAML などで記述することが多い。

▶ ※ CSSはその拡張言語である Sass や LESS などで記述することが多い。

▶ ※ JavaScriptはその拡張言語である CoffeeScript などで記述することが多い。

■データベース技術

▶ ※ RailsがRDBMSの機能をうまくラップしてくれているため、Railsアプリ開発中にSQLを書いたり、RDBMS固有の機能を直接操作したりする場面は非常に少ない。ただし、Railsが提供しているデータベース関連の機能を理解しておく必要がある。

■セキュリティ

▶ ※ Railsアプリケーションに関するセキュリティ上の考慮点について、このドキュメントを参照して必ず理解せよ。

- ▶ ※ Brakeman のようなgemを用いてセキュリティ問題を引き起こすコードが含まれていないか解析するのも有効。

■テストの自動化

- ▶ ※ Railsでは、事前にテスト用のデータをセットアップする（データベースに登録する）必要がある。
- ▶ ※ Railsでは、統合テストでブラウザの操作をシミュレートする必要がある。

■Git/GitHub

- ▶ ※ ブランチやプルリクエストの作成とマージ、コンフリクトが発生した場合の解消方法など、チーム開発で必須となる基本操作を習得しておくのがよい。

■サーバや運用

- ▶ ※ サーバのセットアップ、DNSの管理、デプロイの自動化、サーバの監視、サーバのチューニング、データのバックアップといった知識が必要になる。

■周辺のgem

- ▶ ※ サポートがおろそかになっている gem を使わないようにしよう。
- ▶ ※ gemを導入する前にそのgemを本当に導入すべきかじっくり検討し、「無理に使わなくても何とかなる」「自力で実装することもできる」という場合はgemを使わないのも一つの手である。

- ▶ ※ Brakeman のようなgemを用いてセキュリティ問題を引き起こすコードが含まれていないか解析するのも有効。

■テストの自動化

- ▶ ※ Railsでは、事前にテスト用のデータをセットアップする（データベースに登録する）必要がある。
- ▶ ※ Railsでは、統合テストでブラウザの操作をシミュレートする必要がある。

■Git/GitHub

- ▶ ※ ブランチやプルリクエストの作成とマージ、コンフリクトが発生した場合の解消方法など、チーム開発で必須となる基本操作を習得しておくのがよい。

■サーバや運用

- ▶ ※ サーバのセットアップ、DNSの管理、デプロイの自動化、サーバの監視、サーバのチューニング、データのバックアップといった知識が必要になる。

■周辺のgem

- ▶ ※ サポートがおろそかになっている gem を使わないようにしよう。
- ▶ ※ gemを導入する前にそのgemを本当に導入すべきかじっくり検討し、「無理に使わなくても何とかなる」「自力で実装することもできる」という場合はgemを使わないのも一つの手である。

環境構築編

■環境構築

- ▶ バージョンを確認

■Railsのインストール

- ▶ ☆ 0[1]．Rubyのインストール
- ▶ ☆ 0[2]．gemのインストール時間を短縮する設定に
- ▶ ☆ 1．Railsのインストール

■Railsチュートリアル用にさらに整備

- ▶ ☆ 1．Bundlerをインストール
- ▶ ☆ 2．Yarnをインストール

環境構築編

■環境構築

- ▶ バージョンを確認　　$ rails -v

■Railsのインストール

- ▶ ☆ 0[1]．Rubyのインストール
- ▶ ☆ 0[2]．gemのインストール時間を短縮する設定に
- ▶ ☆ 1．Railsのインストール

■Railsチュートリアル用にさらに整備

- ▶ ☆ 1．Bundlerをインストール
- ▶ ☆ 2．Yarnをインストール

基本編

■思想

- ☆ CoC (Convention over Configuration) - 設定より規約
- ☆ DRY (Don't Repeat Yourself) - 同じことを繰り返さない
- ☆ REST (Representational State Transfer)
- ☆ 自動テスト

■重要な考え方

- ☆ リソースフルルーティング
- ☆ オブジェクト指向
- ☆ MVCアーキテクチャ

■用語

- リソース
- ルーティング

■データ型

- string　　1行の文字列
- text　　複数行の文字列

■初歩的注意

- ※ 基本的には db/ と config/ で設定をいじりつつ app/ の中で Model, View, Controller を作り込んでいく、という流れになる。
- ※ Railsの解説などで「データ」と言う場合、実際にはレコードのことだと思うので、以下の説明で「録（レコード）」と言っているところは元々は「データ」と説明されていたものが多い。
- ※ 以下で `$ rails ~` となっているものは `$ bin/rails ~` にもできる。

■新しくRailsアプリケーションを起動

- ☆ 0．`environment` ディがある前提
- 1．プロジェクトの雛形を作成
- 2．プロジェクトの場所に移動
- ☆ 3．プロジェクトに必要なgemをインストール
- ☆ 4．Webpackerをインストール（Railsチューだけなのか、どうなのか）

■ブラウザでの見え方を確認

基本編

■思想

- ☆ CoC (Convention over Configuration) - 設定より規約
- ☆ DRY (Don't Repeat Yourself) - 同じことを繰り返さない
- ☆ REST (Representational State Transfer)
- ☆ 自動テスト

■重要な考え方

- ☆ リソースフルルーティング
- ☆ オブジェクト指向
- ☆ MVCアーキテクチャ

■用語

- リソース　　URIが指す内容
- ルーティング　　あるURIにアクセスされた時にコントローラのどのメソを実行するか

■データ型

- string　　1行の文字列
- text　　複数行の文字列

■初歩的注意

- ※ 基本的には db/ と config/ で設定をいじりつつ app/ の中で Model, View, Controller を作り込んでいく、という流れになる。
- ※ Railsの解説などで「データ」と言う場合、実際にはレコードのことだと思うので、以下の説明で「録（レコード）」と言っているところは元々は「データ」と説明されていたものが多い。
- ※ 以下で `$ rails ~` となっているものは `$ bin/rails ~` にもできる。

■新しくRailsアプリケーションを起動

- ☆ 0．`environment` ディがある前提
- 1．プロジェクトの雛形を作成　　$ rails new *pjName*　※ `_ver_` 必要かも
- 2．プロジェクトの場所に移動　　$ cd *pj*
- ☆ 3．プロジェクトに必要なgemをインストール
- ☆ 4．Webpackerをインストール（Railsチューだけなのか、どうなのか）

■ブラウザでの見え方を確認

ローカル開発環境の場合

- ▶ ※ ローカルWebサーバーへの接続の許可が必要なのかな？
- ▶ 1．プロジェクトの場所に移動
- ▶ 2．Webアプリサーバ puma を起動
- ▶ 3．ブラウザでの画面を見る

クラウド開発環境の場合

- ▶ ☆ 1．ローカルWebサーバーへの接続を許可
- ▶ 2．プロジェクトの場所に移動
- ▶ 3．Webアプリサーバ puma を起動
- ▶ ☆ 4．ブラウザでの画面を見る

■基本

- ▶ Railsのバージョンを確認
- ▶ 開発用アプリケーションサーバを起動
- ▶ ☆ バックグラウンドでサーバを起動
- ▶ ☆ Railsコマンドについて

■あるページに対するアクセスからの流れ

- 1．あるURIのページに対してあるHTTPメソッドでアクセス
- 2．ルーティングに従って所定のコントローラの所定のアクション（メソッド）を実行
- 3．ビューがアクセス先のページを描画

■DBの操作

- ▶ ※ DBへの接続は設定ファイルに定義された接続情報にもとづく。
- ▶ DBへ接続しコンソール起動
- ▶ DBを作成
- ▶ マイグレーションの内容をDBに反映
- ▶ ※ おそらく、モデルを作成したり変更したりするとマイグレーション (migrationファイル) が更新されるんだろう。そしてその更新をDBに反映させるということなんだろう。
- ▶ DBを削除
- ▶ db/seeds.rb の内容を実行
- ▶ DBを作成し、スキーマとシードデータを読込み
- ▶ DBを削除して再作成し、〃

ローカル開発環境の場合

- ▶ ※ ローカルWebサーバーへの接続の許可が必要なのかな？
- ▶ 1．プロジェクトの場所に移動　$ cd *pj*
- ▶ 2．Webアプリサーバ puma を起動　$ bin/rails s
- ▶ 3．ブラウザでの画面を見る　http://localhost:3000 にアクセス

クラウド開発環境の場合

- ▶ ☆ 1．ローカルWebサーバーへの接続を許可
- ▶ 2．プロジェクトの場所に移動　$ cd *pj*
- ▶ 3．Webアプリサーバ puma を起動　$ rails s
- ▶ ☆ 4．ブラウザでの画面を見る

■基本

- ▶ Railsのバージョンを確認　$ rails -v
- ▶ 開発用アプリケーションサーバを起動　$ rails s
- ▶ ☆ バックグラウンドでサーバを起動
- ▶ ☆ Railsコマンドについて

■あるページに対するアクセスからの流れ

- 1．あるURIのページに対してあるHTTPメソッドでアクセス
- 2．ルーティングに従って所定のコントローラの所定のアクション（メソッド）を実行
- 3．ビューがアクセス先のページを描画

■DBの操作

- ▶ ※ DBへの接続は設定ファイルに定義された接続情報にもとづく。
- ▶ DBへ接続しコンソール起動　$ rails db
- ▶ DBを作成　$ rails db:create
- ▶ マイグレーションの内容をDBに反映　$ rails db:migrate
- ▶ ※ おそらく、モデルを作成したり変更したりするとマイグレーション (migrationファイル) が更新されるんだろう。そしてその更新をDBに反映させるということなんだろう。
- ▶ DBを削除　$ rails db:drop
- ▶ db/seeds.rb の内容を実行　$ rails db:seed
- ▶ DBを作成し、スキーマとシードデータを読込み　$ rails db:setup
- ▶ DBを削除して再作成し、〃　$ rails db:reset

■モデルの操作

- ※ モデル名は必ず**単数形**！
- ※ ここでは `Foo` というモデルにたいする操作をまとめる。

コマンドラインシェルにて

- モデを作成

irb (コンソール)、Rubyファイルの中で

- モデを編集
- 値に制限をつける
- 初期の録たちを作成
- 録を追加
- 全録を確認

DBのコンソール、sqlファイルの中で

- 全録を確認

■コントローラの操作

- ※ コントローラ名は必ず**複数形**！
- ※ ここでは `FoosController` というコントローラにたいする操作をまとめる。

シェルにて

- コンを作成
- ルーティングの設定状況を確認

irb (コンソール) またはRubyファイルの中で

- アクションを追加
- ☆ createアクションの内容
- ルーティングの設定を行う
- ビュ未作成だけど取り急ぎ出力
- リダイレクト

ブラウザにて

- ルーティングの設定状況を確認

■モデルの操作

- ※ モデル名は必ず**単数形**！
- ※ ここでは `Foo` というモデルにたいする操作をまとめる。

コマンドラインシェルにて

- モデを作成　$ rails g model Foo 属性名1:型1 ...

irb (コンソール)、Rubyファイルの中で

- モデを編集　*/pj*/app/models/foo.rb　を編集
- 値に制限をつける　バリデーションを設定　※不当な場合のメッセージもここで設定
- 初期の録たちを作成　*/pj*/db/seeds.rb　を編集
- 録を追加　Foo.create(*attr1*: *value1*, ...)
- 全録を確認　Foo.all

DBのコンソール、sqlファイルの中で

- 全録を確認　select * from foos;

■コントローラの操作

- ※ コントローラ名は必ず**複数形**！
- ※ ここでは `FoosController` というコントローラにたいする操作をまとめる。

シェルにて

- コンを作成　$ rails g controller Foos
- ルーティングの設定状況を確認　$ rails routes　※㋮

irb (コンソール) またはRubyファイルの中で

- アクションを追加　*/pj*/app/controllers/foos_controller.rb の FoosController定義内にメソッドを作成
- ☆ createアクションの内容
- ルーティングの設定を行う　*/pj*/config/routes.rb　を編集
- ビュ未作成だけど取り急ぎ出力　render plain: 式
- リダイレクト　redirect_to *URI*

ブラウザにて

- ルーティングの設定状況を確認　*railsRoot*/rails/info/routes にアクセス　※㋮

■ビューの操作

シェルにて

- コンのhogeアクション
  に対応するビュを作成

HTMLファイル (ERB形式) の中で

- ※ 当然かもしれないが、PHPの埋め込みとだいたい同じだよね。
- <?php ‥ ?> 的な
- <?php echo ‥ ?> 的な
- アプリ共通のHTMLを編集

irb (コンソール)、Rubyファイル、<% %>、<%= %> の中で

- ※ 以下のタグ生成用のヘルパーメソッドでは、追加の引数に `class: oo` や `id: oo` などとして、ほとんどのHTML属性を自由に加えられる。
- リンクタグ
- 画像タグ

- formタグ
- labelタグ
- 送信ボタン
- ☆ ドロップダウンリスト（プルダウンリスト、セレクトボックス）
- ☆ チェックボックス
- ☆ ラジオボタン
- その他のフォーム入力部品タグ
- ※ 実は、列に無い入力フォームも追加できる

CSSファイルの中で

- スタイルシートを編集

実践編

■実践してみよう

- ☆ scaffold で TODOリストをつくってRailsを体験しよう

■ビューの操作

シェルにて

- コンのhogeアクション
  に対応するビュを作成
  /*pj*/app/views/foos/hoge.html　を作成
  ※ ERBエンジンを使いたいなら `hoge.html.erb` に

HTMLファイル (ERB形式) の中で

- ※ 当然かもしれないが、PHPの埋め込みとだいたい同じだよね。
- <?php ‥ ?> 的な　<% ‥ %>
- <?php echo ‥ ?> 的な　<%= ‥ %>
- アプリ共通のHTMLを編集　/*pj*/app/views/layouts/application.html.erb　を編集

irb (コンソール)、Rubyファイル、<% %>、<%= %> の中で

- ※ 以下のタグ生成用のヘルパーメソッドでは、追加の引数に `class: oo` や `id: oo` などとして、ほとんどのHTML属性を自由に加えられる。
- リンクタグ　link_to *content href*
- 画像タグ　/*pj*/app/assets/images/ に画像を置いたうえで　image_tag *imageName*

- formタグ　form_with model: 新しい空の録, local: true do |**f**|↲　‥↲　end
- labelタグ　**f**.label :代入先の列, "*caption*"　※ `"caption"` は省略可
- 送信ボタン　**f**.submit "*caption*"
- ☆ ドロップダウンリスト（プルダウンリスト、セレクトボックス）
- ☆ チェックボックス
- ☆ ラジオボタン
- その他のフォーム入力部品タグ　**f**.*formHelper* :代入先の列
- ※ 実は、列に無い入力フォームも追加できる

CSSファイルの中で

- スタイルシートを編集　/*pj*/app/assets/stylesheets/application.css　を編集

実践編

■実践してみよう

- ☆ scaffold で TODOリストをつくってRailsを体験しよう

▶ ☆ 書籍管理アプリをつくってMVCを理解しよう

▶ ☆ 書籍管理アプリをつくってMVCを理解しよう