データベース環境構築

MySQLとPostgreSQL 大きな違いはないので、好きな方を選べばいい。ただし、現場ではPostgreSQLの方がバグが少ないという理由で使用されている。

PostgreSQLの環境構築 postgresqlのインストール brew install postgresql

encodingをUTF8、localeをja\_JPで初期化 initdb –encoding=UTF-8 –locale=ja\_JP.UTF-8 /usr/local/var/postgres

postgresを起動 postgres -D /usr/local/var/postgres ※postgres -D で起動

Git Gitは、プログラムのソースコードなどの変更履歴を記録・追跡するためのバージョン管理システムです。バージョン管理とは、その名の通り、複数存在するソースコードのバージョンを管理するためのものです。そのバージョン管理を楽にしてくれるのがGitで、これを使用することによって変更履歴をさかのぼってソースコードを元の状態に戻す事ができたり、複数人での共同開発も可能になります。チームでの共同開発をする上で、もはやGitは欠かせない存在となっています。プログラマーというとコードを書いているイメージがあるかと思いますが、それ以前にGitなどのバージョン管理ツールが無ければ仕事になりません。ぜひ、ここで基礎を押さえておきましょう。

Gitにおける開発の流れ まずは基本的な用語から押さえておきましょう。Gitがバージョン管理下におく場所を「リポジトリ」と言います。リポジトリには「ローカルリポジトリ」と「リモートリポジトリ」があり、ローカルリポジトリは自分のパソコン内にあるリポジトリ、リモートリポジトリはネット上にあるリポジトリと覚えておきましょう。リモートリポジトリに関しては、後ほど詳しくご説明いたします。 ユーザが変更の履歴を保存することをコミット（commit）と言います。これはGitの持つ機能の中でも特によく使用するのでぜひ覚えておいてください。コミットは実行した順番で記録されるので、時系列に沿って変更履歴を追う事ができます。そのため、過去にどういう変更を行ったのかを簡単に確認する事ができるのです。また、コミットをする際にはコミットメッセージという形でメモを残す事ができます。例えば、「デザインの変更をしました」とか「データベースへの登録処理を追加しました」といったメモを残す事ができるというわけです。それによって、より変更履歴の理解が容易になります。

複数人での開発を行う際は、この「コミット」と「コミットメッセージ」を利用し、誰がみても理解できるような変更履歴を作成し、任意の履歴に遡ることができるよう心がけましょう。

同時並行で複数の開発を可能にする機能 Gitには、複数の開発を同時並行することができる機能があります。例えば、Aさんはデータの登録を行う処理を実装し、Bさんはデータの表示形式を綺麗にするという実装をするとします。そのとき、2人で同じファイルを編集することになるととても効率が悪くなります。それを解決するための機能が「ブランチ」です。ブランチとは、文字通り開発を枝分かれさせる機能のことで、これを使用することによってAさんとBさんの開発を別々のものとして扱うことができます。

例） Aさん　→ ブランチA Bさん　→ ブランチB

ローカルリポジトリとリモートリポジトリ 前述でGitのバージョン管理下にある場所をリポジトリというというお話をしました。その中でも、ネットを介してどこからでも利用できるようにしたリポジトリのことをリモートリポジトリと言います。それとは逆に、自分のパソコン内にあるリポジトリのことろローカルリポジトリと言います。複数人での開発を行うときは、リモートリポジトリを複数のプログラマーで共有し、そのリモートリポジトリのクローンをそれぞれのプログラマーがローカルリポジトリにコピーして開発を行うというスタイルになります。これによって、１つのプロジェクトを複数人で開発することが可能になるわけです。

SourceTree 通常、GitはターミナルやコマンドプロンプトなどのCUIで操作をするものです。つまり、コマンドだけを打ってバージョン管理を行うということですね。しかし、それだと初心者の方にはハードルが高い。というわけで、視覚的にGitを操作できるようにしたのがSourceTreeというツールです。こちらを使用すれば、初心者の方でも簡単にGitでのバージョン管理を行えるようになります。ちなみに、SourceTreeのようにグラフィックが充実していて視覚的に操作できる画面のことをGUIと言います。今回は、このSourceTreeを利用してGitを操作し、バージョン管理の方法を練習していきましょう。

ローカルリポジトリを作る

コミットする

ブランチを作成する

ブランチをマージする

メソッド

next 繰り返し処理の中で「next」が実行されるとブロック内の「next」以降の処理が行われずに次の繰り返しへ進む

Rubyのクラス 組込ライブラリ Array clear 要素をすべて削除し、配列を空にする。レシーバ自身を変更するメソッド。戻り値はレシーバ自身。

fruits = ["apple", "orange", "banana"]  
fruits.clear  
p fruits  
Enumerator  
next  
「次」のオブジェクトを返す。  
str = "xyz"  
enum = str.each\_byte  
  
str.bytesize.times do  
 puts enum.next  
end  
# => 120  
# 121  
# 122

File join File::SEPARATORを間に入れて文字列を連結

File.join(Rails.root, hoge\_path, hoge\_path, hoge.hoge)  
# => root\_path/hoge\_path/hoge.hoge

ファイルフォーマットライブラリ CSV::Row field ヘッダの名前かインデックスで値を取得。見つからなかった場合はnilを返す。第一引数にヘッダの名前かindex番号を指定する。第二引数にindexを指定し、その番号より後ろにあるヘッダを探す。 row.field(‘field\_name’) row.field(‘field\_name’, 3)

Railsのクラス https://github.com/rails/rails actioncable actionmailer actionpack actionview activejob activemodel activerecord activestorage activesupport

Active Record delete\_all テーブルのレコードを全て削除。関連テーブルの削除は行わない create createを実行すると新しいオブジェクトが返され、さらにデータベースに保存される

import createやsaveよりも効率的にデータを大量投入する事ができる バリデーション抜きでデータを投入する事ができる

使用例 Model.import array\_of\_models Model.import column\_names, array\_of\_models Model.import array\_of\_hash\_objects Model.import column\_names, array\_of\_hash\_objects Model.import column\_names, array\_of\_values Model.import column\_names, array\_of\_values, options

module moduleとは、オブジェクトを生成することはできないがメソッドを格納できるもの モジュールは多重継承を行うことができる Rubyではクラスを多重継承することができないためモジュールでそれを補う モジュールの中で定義したメソッドには以下のような使い方がある 定義したメソッドを直接実行する クラスの中にインクルードしてメソッドを実行する

クラスにモジュールをインクルードする場合の手順は以下のようになる app/libにモジュールファイルを作成 ファイルの命名規則 ファイル名は小文字 単語の区切りは\_ モジュール名はキャメルケース モジュールの定義をする

module モジュール名  
  
end

# classのなかにincludeを記述する  
class hoge  
 include huga  
end

## rake task

rakeとは、Rubyで書かれたコードをタスクとして作成しておき必要に応じて呼び出して実行することができる機能 rake taskを実行するとデータベースへのデータの大量投入も楽になります。

rakeタスクファイルを生成する

rails g task task\_name

lib/taskに以下のファイルが作成される

task\_name.rake

中身は以下のようになっている

namespace :task\_name do  
end

中に処理を追加する

task :processing\_name => :environment do |task|  
 # rake 'task\_name:processing\_name'  
 行いたい処理を追記  
end

rakeタスクの確認をする

rake -vT

表示されたタスクを確認し実行する

## CSVでのデータの大量投入

開発の現場では大量のデータを取り扱うため、ExcelでCSVデータを作成し、それをデータベースに一気に投入するという方法がよく使用されています。汎用性の高い技術なので、ぜひ自分で実装できるようにしましょう。

新規に以下の名前のアプリを作成しましょう。

rails new csv\_data

次に、データ投入のコードを記述するためのmodelを作成します。

rails g model data\_test

modelと同時に作成されたマイグレーションファイルを、以下のように編集しましょう。今回は、name,age,addressの３つカラムを作成し、データ投入を行います。そのため、nameをstring、ageをinteger、addressをstringで作成します。

class CreateDataTests < ActiveRecord::Migration[5.2]  
 def change  
 create\_table :data\_tests do |t|  
 t.string :name  
 t.integer :age  
 t.string :address  
 t.timestamps  
 end  
 end  
end

マイグレーションファイルを実行し、データベースにテーブルを作成します。

bundle exec rake db:migrate

models/application\_record.rbにCSVデータをインポートするためのコードを追記します。

# この記述がないとNameError: uninitialized constant DataTest::CSVがでる  
require 'csv'  
  
models/data\_test.rbに以下を追記  
def self.import(path)  
 CSV.foreach(path, headers: true) do |row|  
 self.find\_or\_create\_by(  
 name: row["name"],  
 kana: row["age"],  
 address: row["address"]  
 )  
 end  
end

次に、データを投入するためのCSVファイルを保管するディレクトリを作成しましょう。dbディレクトリの中にcsv\_dataディレクトリを作成し、その中にCSVファイル（名前：csv\_data.csv）を用意します。CSVファイルは初回配布データからコピー&ペーストで配置しましょう。CSVファイルには以下のような記述がされています。

name,age,address  
hogehoge1,hogehoge1,hogehoge1  
hogehoge2,hogehoge2,hogehoge2  
hogehoge3,hogehoge3,hogehoge3  
hogehoge4,hogehoge4,hogehoge4  
hogehoge5,hogehoge5,hogehoge5

以下のコマンドでコンソールを立ち上げる

rails c

以下のコマンドを実行

DataTest.import('db/csv\_data/csv\_data.csv')

## Ajaxの導入

Ajaxとは「Asynchronous JavaScript ＋ XML」の略で、JavaScriptとXMLを使って非同期にサーバとの間の通信を行う方法(非同期通信)のことです。非同期通信とは、画面の一部だけを更新するような通信で、画面全体を更新することなく、一部だけをユーザのリクエストに応じて変化させることができます。非同期通信ではレスポンスを待っている間も他の操作を行うことができるので、ユーザーのストレスを大幅に軽減することができます。

今回作成するサンプル 今回は、人の名前をAjax通信でデータベースに登録し、一覧表示させるというシンプルなアプリを作成します。このサンプルを通して、Ajax通信でのデータ登録機能を実装する方法を覚えましょう。余裕がある人は削除機能の実装にもチャレンジしてみてください。

新規に以下の名前のアプリを作成しましょう。

rails new ajax\_sample

ajax\_sampleのディレクトリに移動します。

cd ajax\_sample

ajax\_sampleのデータベースを作成します。

bundle exec rake db:create

サーバを起動し、正常に初期画面が表示されるか確認しましょう。

rails s

次に、データの投入対象となるusersテーブルに紐づくmodelを作成します。

rails g model user

modelと同時に作成されたマイグレーションファイルを、以下のように編集しましょう。今回は、nameカラムを作成し、Ajax通信を利用してデータを登録します。

class CreateUsers < ActiveRecord::Migration[5.2]  
 def change  
 create\_table :data\_tests do |t|  
 t.string :name  
 t.timestamps  
 end  
 end  
end

マイグレーションファイルを実行し、データベースにusersテーブルを作成しましょう。

bundle exec rake db:migrate

今回はAjax通信をRailsに導入するため、jQueryを導入します。従って、Gemファイルに以下のコードを追記しましょう。

gem 'rails-ujs'  
gem 'jquery-rails'

追記したGemをインストールします。

bundle install

読み込み対象となるJSファイルのjqueryを指定するため、application.jsに以下の内容を追記します。

//= require jquery

routes.rbに以下のルートを記述します。今回は、getでもpostでも同じアクションが指定されるようにしています。その理由は、Ajax通信（非同期通信）を行うため、ページを遷移する必要がなくpost通信でリクエストされた場合でもindex.html.erbのみを表示するためです。

root to: 'users#index'  
get '/users', to: 'users#index'  
post '/users', to: 'users#index'

ルートでusersコントローラを指定したので、usersコントローラを作成します。以下のコマンドを実行しましょう。

rails g controller users index

usersコントローラにindexアクションを追記し、その中に以下の処理を記述しましょう。

def index  
 @users = User.all  
end

まずはuserを一覧表示するための画面を作成しましょう

<h1>User一覧</h1>  
<% @users.each do |user| %>  
 <p>お名前　<%= user.name %></p>  
<% end %>

次に、投稿フォームを作成するための準備をします。先ほど編集したindex.html.erbにコードを追記しましょう。

<%= link\_to "ユーザ登録", users\_new\_path, remote: true %>  
<div id="users-form"></div>  
<div id="users\_index">  
 <%= render 'index' %>  
</div>

今回は、index.html.erbで部分テンプレートを指定していますので、対応するファイルを作成します。usersディレクトリに\_index.html.erbを作成し、以下のコードを記述しましょう。

<h1>User一覧</h1>  
<% @users.each do |user| %>  
 <p>お名前　<%= user.name %></p>  
<% end %>

次に、新規登録画面に遷移するルートを設定しましょう。

root to: 'users#index'  
get '/users', to: 'users#index'  
get '/users/new', to: 'users#new'

新規登録フォームのroutesで指定されたコントローラのアクションはnewです。そのため、コントローラに以下のアクションを追記しましょう。

def new  
 @user = User.new  
 respond\_to do |format|  
 format.html  
 format.js  
 end  
end

新規登録フォームを表示させるためのjsファイルを作成します。コントローラのnewアクションが実行された時に実行されるjsファイルを作成したいので、usersディレクトリにnew.js.htmlを作成します。そのファイルに以下のコードを記述しましょう。

$('#users-form').html("<%= j (render 'form') %>");  
$('#users-form').fadeIn(800);

「render ‘form’」で表示させるためのファイルを作成しましょう。名前は「*form.html.erb」です。*(アンダースコア)を入れることによって部分テンプレートとして機能し、jsファイルが実行されたタイミングで\_form.html.erbが「

」の部分に表示されるようになります。 <%= form\_for @user, method: :post, remote: true do |f| %> <%= f.label “名前” %> <%= f.text\_field :name %> <%= f.submit %> <% end %>

次に、フォームへデータを入力後のデータベースへの登録処理を実装しましょう。まずはroutesから定義します。 root to: ‘users#index’ get ‘/users’, to: ‘users#index’ get ‘/users/new’, to: ‘users#new’ post ‘/users’, to: ‘users#create’

今回指定するcontrollerのアクションはcreateなので、controllerにcreateアクションを定義しましょう。アクション内に書いてあるrespond\_toの部分は、「ユーザーがhtmlをリクエストしているならhtmlを表示し、jsをリクエストしているならjsファイルを実行する」という意味になります def create @users = User.all @user = User.new(users\_params) respond\_to do |format| if @user.save format.html format.js else @message = @user.errors.full\_messages format.js end end end

次にcreate.js.erbファイルを作成し、以下のコードを記述しましょう。 $(‘#users\_index’).html(“<%= j (render ‘index’) %>”); $(‘#users-form’).fadeOut(600);

このコードにより、users\_indexのタグがある場所に\_index.html.erbの部分テンプレートを表示させ、users-formのタグがある場所を非表示にすることができます。

simple\_formの導入 form\_tagやform\_forのようにフォームを作成するためのヘルパーメソッド モデルオブジェクトに対応したフォームを作成することができるため、データベースへのデータ投入も行いやすい

gem ‘simple\_form’ bundle install

simple\_formでの登録 <%= simple\_form\_for @users, :url => {:action => :create} do |f| %> <%= f.collection\_check\_boxes :programming, [[“Java”, “Java”],[“Ruby”, “Ruby”],[“PHP”, “PHP”],[“Python”, “Python”]], :first, :last %> <%= f.button :submit, “Sign UP！”, :class=>“btn btn-primary btn-lg center-block” %> <% end %>

strong parameter で配列を許可 def users\_params 　　params.require(:user).permit(:programming => []) end

カラムのデータ型を配列が登録できるように変更 rails g migration change\_datatype\_programming\_of\_users(テーブル名) migration ファイルの中身を編集 class ChangeDatatypeProgrammingOfUsers < ActiveRecord::Migration def change change\_column :users, :programming, :text, array: true end end

Herokuにアップロードするときは以下のやり方じゃないとエラーになる class ChangeDatatypeProgrammingOfUsers < ActiveRecord::Migration def change change\_column :users, :programming, ‘text USING CAST(programming AS text[])’ end end

マイグレーションを実行する bundle exec rake db:migrate

model でのカラムの serialize 化 serialize :programming

コントローラでレコードを全件取得 def index @users = User.all end

取り出した配列を view で配列のメソッドを使用して表示 @user.programming.delete\_if(&:empty?).join(“,”)

Rakeタスク アプリケーションを起動せずに行いたい処理をCUIから実行できます。CSVデータのインポートをする際にこの機能がよく利用されます。 ※puts の色をわかりやすくするために gem ‘colorize’ を導入しています

# namespace\_name, task\_nameには任意の名前を入力してください。

namespace :(namespace\_name) do task (task\_name): :environment do end end

namespaceは名前空間と呼ばれるもので、これを利用することによってタスクをグループ分けする事ができるようになります。

実行するときは、ターミナルから以下のようにコマンドを入力します。 rake namespace\_name:task\_name

では、実際にtaskを定義する事ができるか確かめてみましょう CSVデータ投入用タスクのサンプルコード require ‘csv’

namespace :import do desc “Import useers from csv”

task users: :environment do path = File.join Rails.root, “db/csv/csv\_data.csv” puts “path: #{path}”.red list = [] CSV.foreach(path, headers: true) do |row| list << { name: row[“name”], age: row[“age”], address: row[“address”], gender: row[“gender”], program: row[“program”], skils: row[“skils”], } end puts “start to create users data”.red begin User.create!(list) puts “completed!!”.green rescue puts “raised error”.red end end end

gem gemとは、Ruby用のパッケージ管理ツールです。このgemのおかげで、Ruby用ライブラリのインストール、アンインストールなどを簡単に管理することができます。

sitemap\_generator

以下のgemを追加 gem ‘sitemap\_generator’

sitemap.rbを作成 gem install sitemap\_generator

用途に合わせて以下のいずれかのコマンドを実行 rake sitemap:create => sitemapの作成を行いたい時 rake sitemap:refresh => sitemapを作成してサーチエンジンにsitemapを更新したことを通知したい時 rake sitemap:refresh:no\_ping => sitemapを作成し、サーチエンジンにsitemapの更新を通知したくない時 rake sitemap:clean => sitemapを削除したい時

ransack ransackはrails用の検索機能を実装するためのgem

以下のgemを追加 gem ‘ransack’ bundle install

modelのリレーションを追加 user model has\_many :comments comment model belongs\_to :user

userモデルからcommentを検索する場合 attr\_accessor :comment def execute User.ransack(comments\_eq: @comment) end

検索したいデータをmodelオブジェクトを生成する際に渡してあげる @user = User.new(comment) @user.execute # データ検索ができる。commentにデータがない場合はall検索となる

Bootstrap4の導入 Twitter社が開発したフロントエンドのフレームワークで、CSSを書かずにおしゃれでスマホなどの幅の違う画面にも対応したデザインができます。このBootstrapを取り入れる事でフロントエンドの学習効率が飛躍的に上がります。

gemファイルに以下のコードを追記しましょう。 gem ‘bootstrap’, ‘~> 4.1.1’ gem ‘jquery-rails’

gemをインストール bundle install

sprockets-railsが2.3.2以上であることを確認 bundle show |fgrep sprockets-rails

CSSファイルの拡張子を変更 application.css => application.scss

application.scssに以下のコードを追記 @import “bootstrap”;

application.scssの以下のコードを削除 \*= require\_tree .

applicaiton.jsに以下のコードを追記 //= require jquery3 //= require popper //= require bootstrap-sprockets

レスポンシブに対応させるため以下のコードを application.html.erbのheadタグの中に記述

以下のコードをhtml.erbに記述し、Bootstrapが適用されているか確かめる Tooltip on top

activeadmin 管理者画面を簡単に作成するためのGem データの作成、更新、削除を簡単に管理できるようになる このGemを利用することによって、ユーザー用、管理者用という風に１つのアプリケーションの画面を使い分けられるようになる

以下のgemをGemfileに追記 gem ‘activeadmin’ gem ‘devise’

gemをインストール bundle install

activeadminをインストール rails generate active\_admin:install

マイグレートの実行 rake db:migrate

初期ユーザデータを作成するためシードファイルを実行 rake db:seed ※simple form を使用しているとエラーが出る：次のコマンドで解決：rails rails generate simple\_form:install

サーバを起動 rails server

以下のURLにアクセス http://localhost:3000/admin

以下のユーザでアクセス User: admin@example.com Password: password

モデルを作成（必要な場合） rails generate active\_admin:resource user Userが管理画面に追加されていることを確認

管理画面を日本語化する gemに以下を追加する gem ‘rails-i18n’

config/localsにja.ymlを追加し、必要に応じて設定したいラベルの日本語を追記する ja: activerecord: models: attributes: user: name: 名前 email: メールアドレス simple\_form: labels: user: name: ‘お名前’ age: ‘年齢’ address: ‘住所’ gender: ‘性別’ program: ‘勉強中の言語’ skils: ‘スキル一覧’ curriculum: course: ‘コース名’ hour: ‘学習時間’ price: ‘受講料’

管理画面の編集、削除フォームを日本語化する config/application.rbに以下の記述を追記 config.i18n.default\_locale = :ja config.i18n.load\_path += Dir[Rails.root.join(‘config’, ‘locales’, ’\*\*‘,’\*.{rb,yml}’).to\_s]（locales

直下にja.ymlを置く場合この記述は必要ない） config/locales/models/ja.ymlを作成し、以下の内容を記述する ja: activerecord: attributes: モデル名: モデルのカラム名: 設定したい値

ja: activerecord: attributes: doctor: waiting\_weeks: 待機週間（数値）

decorator コントローラから受け取った値に対し、表示形式を整えたり、複雑な処理を行ってviewに表示したい際に使用する

gemの導入 gem ‘active\_decorator’ bundle install

decorator のファイルを作成 rails g decorator user(model名) app/decorators にあるdecorator を編集

module UserDecorator def programming\_status programming.delete\_if(&:empty?).join(“,”) end end

view でメソッドを呼び出す <%= user.programming\_status %>

gretel 簡単にパンくずリストを作成するためのGem パンくずリストはSEOに効果があると言われています

gemファイルを編集 gem ‘gretel’

gemをインストール bundle install

以下のコマンドを実行し、「config/breadcrumbs.rb」が作成されていることを確認 bundle exec rails generate gretel:install

「breadcrumbs.rb」を編集し、リストを作成するときの設定を行う crumb :root do link ‘Home’, root\_path end

crumb :users do link ‘ユーザー一覧’, users\_path end

crumb :user do |user| link “@#{user.nickname}”, user\_path(user) parent :users end

「app/views/layouts/application.html.erb」のパンくずリストを表示したい部分に以下のコードを記述 <%= breadcrumbs pretext: “You are here:”, separator: " › " %>

「app/views/users/index.html.erb」にユーザ一覧表示のパンくずリストを追加（表示したい箇所に記述） <% breadcrumb :users %> 「app/views/users/show.html.erb」にユーザ詳細表示のパンくずリストを追加（表示したい箇所に記述） <% breadcrumb :user, @user %>

carrierwave, rmagick gemをインストール gem ‘carrierwave’ gem ‘rmagick’, require: ‘RMagick’

gemをインストール(上記以外の場合) gem ‘carrierwave’, :github => ‘satoruk/carrierwave’ , :ref => ‘43179f94d6a4e62f69e812f5082d6447c9138480’ gem ‘rmagick’, require: false bundle install

上記でエラーが出る場合RMagickを6系から7系に切り替える brew unlink imagemagick brew uninstall imagemagick brew install imagemagick@6 brew link imagemagick@6 –force

マイグレーションファイルの作成 rails g migration AddImageToUser image:text bundle exec rake db:migrate

アップローダを作成 rails g uploader image

アップローダファイルの中身を書き換え(uploaders/image\_uploader.rb) class ImageUploader < CarrierWave::Uploader::Base include CarrierWave::RMagick storage :file process convert: ‘jpg’ # 保存するディレクトリ名 def store\_dir “uploads/#{model.class.to\_s.underscore}/#{mounted\_as}/#{model.id}” end # thumb バージョン(width 400px x height 200px) version :thumb do process :resize\_to\_fit => [400, 200] end # 許可する画像の拡張子 def extension\_white\_list %W[jpg jpeg gif png] end # 変換したファイルのファイル名の規則 def filename “#{Time.zone.now.strftime(‘%Y%m%d%H%M%S’)}.jpg” if original\_filename.present? end end

画像をアップするモデルクラスに以下の記述を追加 mount\_uploader :image, ImageUploader

ファイルをアップロードするフォームを作成（form\_for , simple\_form\_forに記述） <%= f.file\_field :image %>

画像を表示させる記述（form\_for , simple\_form\_forに記述） <%= image\_tag user.image\_url(:thumb) %>

spreadsheet（未実装） excelのデータをインポートするGemです。.xlsx, .xlsmは扱えません。excelデータの読み込み、データベースへの大量投入が可能になります。

gem ‘spreadsheet’

excel.rbにエンコーディングの設定をする Spreadsheet.client\_encoding = “UTF-8”

エクセルファイルを新規作成して保存するときのexcel.rbの設定 Spreadsheet.client\_encoding = “UTF-8” book = Spreadsheet::Workbook.new sheet1 = book.create\_worksheet sheet1.name = “test” # 処理書く book.write “/path/excel.xls”

エクセルファイルを開いて別名で保存

logger loggerはログを出力するために、Railsにあらかじめ用意されている機能 バグの検出が容易になるので便利

config.logger = Logger.new(STDOUT) config.logger = Logger.new(‘log/development.log’)

Logger.new関数では第1引数にログを保存する場所を指定 STDOUTは標準出力にログを出力する場合の引数

loggerでは、ログにレベルを設定できる

unknown(常にログとして出力される必要がある未知のメッセージ) fatal(プログラムをクラッシュさせる制御不可能なエラー) error(制御可能なエラー) warn(警告) info(システム管理についての一般的、または役に立つ情報) debug(開発者のための低いレベルの情報)

出力の優先順位 unknown>fatal>error>warn>info>debug

ログレベルの設定 config.log\_level = :unknown config.log\_level = :fatal config.log\_level = :error config.log\_level = :warn config.log\_level = :info config.log\_level = :debug

config.log\_level = :warn にするとinfo,debugは出力されなくなる

ログレベルは以下のファイルで設定できる ・config/application.rb ・config/evironments/development.rb ・config/evironments/production.rb ・config/evironments/test.rb

上記のファイル以外でログレベルを設定したい場合は以下のコードで可能

0(debug) 1(info) 2(warn) 3(error) 4(fatal) 5(unknown)

RailsでのSEO対策 リッチカードの導入 リッチカードを使用すると、テキストよりも視覚に訴える形で検索結果を表示する事ができ、結果としてSEOに強くする事ができます。リッチカードをWebサイトに導入するには、構造化データをHTMLの中に記述する必要があります。

構造化データの例

構造化データのサンプル集 こちらのサイトに構造化データのサンプル集があります。0から自分で書く必要はありません。 https://developers.google.com/search/docs/data-types/corporate-contact

コードを書く場所 構造化データのコードはheadタグの中に記述する。

Railsへの導入手順 publicの中にrich-cardで表示するための画像を置くディレクトリを作成します。 puclic/rich-card

rich-cardディレクトリの中にlogoの部分に表示したい画像を配置します logo.png

構造化データを作成する

例外処理 例外が発生してもシステムを止めないようにする

基本的な書き方 begin   例外が発生する可能性のある処理 rescue   例外発生時に行う処理 end

意図的に例外を発生させるときはraiseを利用する raise ArgumentError, “#{path.to\_s} was not existed”

例外の種類

Exception 全ての例外クラスの親クラス StandardError 通常のプログラムで発生する可能性の高い例外クラスをまとめたクラス ArgumentError 引数のあっていない時や、数はあっていても期待される値ではない時 NameError 未定義のローカル変数や定数を使用した時に発生

例外クラスについてさらに深く知りたいならこちらを参照 https://docs.ruby-lang.org/ja/2.5.0/library/\_builtin.html

組み込みライブラリ このライブラリに組み込まれているクラスやモジュールはrequireを書かなくても使用できる

現場で利用されているGem一覧 gem ‘dotenv-rails’, groups: [:development, :test]

gem ‘rails’, ‘~> 5.1.0’ gem ‘puma’ gem ‘bootstrap-sass’ gem ‘sprockets’ gem ‘sprockets-rails’ gem ‘bcrypt’ gem ‘pg’ gem ‘nokogiri’

group :development, :test do gem ‘rspec-rails’, ‘~> 3.6.1’ gem ‘guard’ gem ‘guard-rspec’ gem ‘spork’, ‘~> 1.0rc’ gem ‘guard-spork’ gem ‘childprocess’ gem ‘rb-readline’ gem ‘bullet’ gem ‘pry-rails’ gem ‘pry-doc’ gem ‘pry-byebug’ gem ‘pry-stack\_explorer’ gem ‘rails-controller-testing’ gem ‘timecop’ end

gem “exception\_notification” gem ‘slack-notifier’

group :test do gem ‘selenium-webdriver’ gem ‘capybara’ gem ‘factory\_girl\_rails’ gem ‘cucumber-rails’, :require => false gem ‘minitest’ gem ‘faker’ end

# gem ‘sass-rails’, ‘>=4.0.2’

gem ‘uglifier’ gem ‘coffee-rails’ gem ‘jquery-rails’ gem ‘jbuilder’

group :doc do gem ‘sdoc’, require: false end

group :production do gem ‘rails\_12factor’ end

gem ‘ransack’ gem ‘devise’ gem ‘devise-i18n’ gem ‘omniauth’ gem ‘omniauth-facebook’ gem ‘omniauth-google-oauth2’

gem ‘meta-tags’ gem ‘sitemap\_generator’

gem ‘activeadmin’ gem ‘active\_admin\_import’, github: ‘activeadmin-plugins/active\_admin\_import’ gem ‘cancancan’

gem ‘payjp’

gem ‘activerecord-postgis-adapter’ gem ‘geocoder’

gem ‘activerecord-session\_store’

gem ‘kaminari’ gem ‘gretel’

gem ‘httpclient’ gem “httparty” # for image uploader with AWS S3 gem ‘carrierwave’ gem ‘fog’

group :development, :test, :api do gemspec path: ‘vendor/engines/api’ gemspec path: ‘vendor/engines/staff’ gem “awesome\_print” # use ap {Model} in rails console

gem “letter\_opener” gem “letter\_opener\_web” end

# for cron job scheduler

gem ‘whenever’, require: false

# for bulk insert

gem ‘activerecord-import’, ‘~> 0.17.0’

# for get japanese holiday

gem ‘holiday\_jp’

# for auto link in interview

gem ‘rails\_autolink’

# for elb health check

gem ‘health\_check’

# for react

gem ‘webpacker’, ‘~> 3.0’ gem ‘react-rails’ gem ‘react-bootstrap-rails’

# for xls file import

gem ‘spreadsheet’

# for font awesome

gem ‘font-awesome-rails’

gem ‘rack-cors’, :require => ‘rack/cors’

# for image processing

gem ‘mini\_magick’

# for accessing aws

gem ‘aws-sdk-dynamodb’