■ Ruby on Rails はじめました

Getting Started with Rails を．ホスト OS は Ubuntu 20.04.

準備

Ruby と SQLite3 が必要，ということで，

```
$ sudo apt install ruby-full
$ sudo apt install sqlite3-doc
$ sudo apt install ruby-bundler
```

でずばっと．bundle はあれこれ必要になるのでいれとく．

```
$ ruby --version
ruby 2.7.0p0 (2019-12-25 revision 647ee6f091) [x86_64-linux-gnu]
$ sqlite3 --version
3.31.1 2020-01-27 19:55:54 3bfa9cc97da10598521b342961df8f5f68c7388fa117345eeb516eaa837balt1
```

が入った．で，gem で rails をインストール．

```
$ sudo gem install
```

やってみる

ガイド通りに

```
rails new blog
```

blog/ 以下にあれこれできている．

```
cd blog
./bin/rails server
```

でなんか起動した．

Unicorn/Nginx 経由でアクセスできるようにする

【Rails】Web サーバー「Unicorn」の基本情報と実装方法を参考に
Gemfile に

```
gem "unicorm"
```

を追加．PID とソケット，ログの格納用ディレクトリを作る．

```
mkdir -p tmp/pids tmp/sockets log
```

で，設定ファイル config/unicorn.rb を丸っと作成．

```
# Rails アプリのルート
rails_root = File.expand_path('../../', __FILE__)

# Gemfile の場所
ENV['BUNDLE_GEMFILE'] = rails_root + "/Gemfile"

# Unicorn の設定
worker_processes  2
timeout           15
working_directory rails_root
pid               File.expand_path 'tmp/pids/unicorn.pid', rails_root
listen            File.expand_path 'tmp/sockets/.unicorn.sock', rails_root
stdout_path       File.expand_path 'log/unicorn.log', rails_root
stderr_path       File.expand_path 'log/unicorn.log', rails_root
preload_app       true

# フォークが行われる前の処理
before_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.connection.disconnect!
  old_pid = "#{server.config[:pid]}.oldbin"
  if old_pid != server.pid
    begin
      Process.kill "QUIT", File.read(old_pid).to_i
    rescue Errno::ENOENT, Errno::ESRCH
    end
  end
end

# フォークが行われた後の処理
after_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.establish_connection
end

# フォークが行われる前の処理
before_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.connection.disconnect!
end

# フォークが行われた後の処理
after_fork do |server, worker|
  defined?(ActiveRecord::Base) and ActiveRecord::Base.establish_connection
end

# フォークが行われる前の処理
before_fork do |server, worker|
  old_pid = "#{server.config[:pid]}.oldbin"
  if old_pid != server.pid
    begin
      Process.kill "QUIT", File.read(old_pid).to_i
    rescue Errno::ENOENT, Errno::ESRCH
    end
  end
end
```

これで,

```
$ bundle exec unicorn -c config/unicorn.rb
```

とすると tmp/pids/unicorn.pid が作成されて PID が格納されて，終了時に削除される.

Nginx 側にエントリを追加するために site-available/hoge を編集.
Nginx から unicorn にアクセスできるようにアップストリームの定義を追加して,

```
upstream unicorn {
        server unix://home/miyo/blog/tmp/sockets/.unicorn.sock;
}
```

/ へのアクセスを定義したアップストリームに振れるように

```
    location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $http_host;
        proxy_pass http://unicorn;
    }
```

とか．で，

```
sudo nginx -t
sudo systemctl restart nginx.service
```

で再起動．あとは，

```
bundle exec unicorn -c config/unicorn.rb
```

で Unicorn 起動した後でブラウザから，アクセスする．
うまく Nginx → unicorn の設定ができていれば，
Blocked host: ホスト名って感じのエラーがでる．
調べてみると，DNS リバインディング攻撃を防ぐため，らしい．

```
config.hosts << "ホスト名"
```

を追加せよ，とあるので，config/environments/development.rb の

```
Rails.application.configure do
 ...
end
```

の末尾に，

```
config.hosts << "ホスト名"
```

を追加．あらためて

```
bundle exec unicorn -c config/unicorn.rb
```

で起動してブラウザからアクセスすると，なんかそれっぽいページが表示される．

Say Hello

MVC の V と C を作って Hello World しよう，という内容．
config/routes.rb を変更

```
Rails.application.routes.draw do
  # Define your application routes per the DSL in https://guides.rubyonrails.org/routing.html

  # Defines the root path route ("/")
  root "articles#index"
  get "/articles", to: "articles#index"

end
```

で,

```
bin/rails generate controller Articles index --skip-routes
```

として controller と view を生成.

- app/controllers/articles_controller.rb
- app/views/articles/index.html.erb
- test/controllers/articles_controller_test.rb
- app/helpers/articles_helper.rb

が生成された.
app/views/articles/index.html.erb の 中身をガイドに従って編集して,
https:// どこか /articles/ にアクセスしたら，index.html.erb の表示がみえた.
config/routes.rb をもう一度変更して,

```
Rails.application.routes.draw do
  root "articles#index"
  get "/articles", to: "articles#index"
end
```

としたら，https:// どこか / にアクセスして作成した index.html.erb の表示がみえた.

Model も作る

ガイド通りコマンドを実行してみる.

```
bin/rails generate model Article title:string body:text
```

ファイルが生成された.

- db/migrate/20221002090126_create_articles.rb
- app/models/article.rb
- test/models/article_test.rb
- test/fixtures/articles.yml

Articleっていう名前で，string 型の title と body をもったモデルを作る，ということのよう.
db/migrate/20221002090126_create_articles.rb をみると DB を作る create_table が定義されているで

```
bin/rails db:migrate
```

を実行.

```
bin/rails console
```

で irb 使って DB をいじる ... ようだ．とりあえず言われるがままにやってみる.

```
article = Article.new(title: "Hello Rails", body: "I am on Rails!")
article.save
```

Article.find(1) とか Article.all とかするとエントリをひける．
裏では，

```
SELECT "articles".* FROM "articles" WHERE "articles"."id" = ? LIMIT ?  [["id", 1], ["LIMIT", 1]]
```

とかって SQL が走っているようだ．
C と V を変更してブラウザからのアクセスで利用できるようにする．
app/controllers/articles_controller.rb を

```
class ArticlesController < ApplicationController
  def index
    @articles = Article.all
  end
end
```

と編集し，app/views/articles/index.html.erb を

```
<h1>Articles</h1>
<ul>
  <% @articles.each do |article| %>
    <li>
      <%= article.title %>
    </li>
  <% end %>
</ul>
```

にする．erb の @articles は，controller.rb の @articles に対応している．
Ruby の return value が欲しくないときは <% %> を，欲しいときには <%= %> を使うらしい．

CRUD の Read を試す

route を追加する．config/routes.rb をアップデート

```
Rails.application.routes.draw do
  root "articles#index"
  get "/articles", to: "articles#index"
  get "/articles/:id", to: "articles#show"
end
```

:id は route のパラメタ．
で，app/controllers/articls_controller.rb にコントローラを追加．
route に追加した show と :id がどう使われるか見てとれる．

```
 def show
   @article = Article.find(params[:id])
 end
```

また，app/views/articles/show.html.erb を作成

```
<h1><%= @article.title %></h1>
<p><%= @article.body %></p>
```

Web ブラウザで / にアクセスするとテーブルに登録された一覧と show へのリンクが.
で，リンク先の /articles/1 などにアクセスすると，show の定義通り title と body が.

CRUD 関連の便利な記法

CRUD 関連の route をひとつひとつ定義するかわりに resources が使えるらしい.
config/routes.rb をアップデート

```
Rails.application.routes.draw do
  root "articles#index"
  resources :articles
end
```

bin/rails routes で確認すると，articles に関連して，あれこれ定義されているのがわかる.

```
root         GET    /                           articles#index
articles     GET    /articles(.:format)         articles#index
             POST   /articles(.:format)         articles#create
new_article  GET    /articles/new(.:format)     articles#new
edit_article GET    /articles/:id/edit(.:format) articles#edit
article      GET    /articles/:id(.:format)     articles#show
             PATCH  /articles/:id(.:format)     articles#update
             PUT    /articles/:id(.:format)     articles#update
             DELETE /articles/:id(.:format)     articles#destroy
```

index.html.erb も a タグを自分で書くんじゃなくて，簡略化できる

```
<h1>Articles</h1>
<ul>
  <% @articles.each do |article| %>
    <li>
        <%= link_to article.title, article %>
    </li>
  <% end %>
</ul>
```

CRUD の Create

サンプルまま作業．ただ，そのままだと，

```
 ActionController::InvalidAuthenticityToken (HTTP Origin header (https://****) didn't match
request.base_url (http://****)):
```

なエラーがでて POST がはじかれていた.
これは Rails 側ではなくて，Nginx で https をほどいているのが問題.
というわけで，site-available/hoge の location / {...} に，

```
proxy_set_header origin 'http://****';
```

を追加.

hoge.html.erb で，他の .html.erb ファイルを読みこむことができる．

```
<%= render "form", article: @article %>
```

とかすると，_form.html.erb と，先頭に '_' が付いたファイルが読み込まれる，ようだ．