第 3 章

トップページの作成

前章で整理・整頓した NanoPlanner のソースコードに対して、トップページを表示する機能を追加していきます。また、SASS、Bootstrap、Font Awesome などのライブラリを導入しフロントエンド開発の基盤を整えます。

3.1 RAVT

『初級①』の第 6 章で、「RAVT」という私の造語を紹介しました。経路（route）、アクション（action）、ビューモジュール（view）、テンプレート（template）の頭文字を並べた略語です。Phoenix アプリに新しい仕様を追加する時には、この順に作っていくのが私のお勧めです。

この章の目的は、トップページを表示することです。この機能を受け持つコントローラは `top`、アクションは `index` としましょう。

> 本書では、「`top` コントローラの `index` アクション」のことを「`top#index` アクション」と呼ぶことにします。

RAVT の手順に沿うとすれば、作業は次のように進んでいくことになります。

- URL パス `/` から `top#index` アクションへの経路を設定する。
- `top#index` アクションを実装する。
- `top` コントローラのビューモジュールを作成する。

- top#index アクションのテンプレートを作成する。

経路の設定

まず、第 1 の手順です。web ディレクトリの router.ex を次のように書き換えます。

web/router.ex

```
  :
12    scope "/", NanoPlanner do
13      pipe_through :browser
14 +
15 +    get "/", TopController, :index
16    end
  :
```

URL パス / から top#index アクションへの経路を設定しています。

アクションの作成

第 2 の手順は、top#index アクションの実装です。web/controllers ディレクトリに新規ファイル top_controller.ex を次の内容で作成してください。

web/controllers/top_controller.ex (New)

```
1   defmodule NanoPlanner.TopController do
2     use NanoPlanner.Web, :controller
3
4     def index(conn, _params) do
5       render conn, "index.html"
6     end
7   end
```

ほぼ同じ内容のソースコードについて『初級①』第 7 章で詳しく解説しましたので、ここでは改めて説明しません。要するに、top#index アクションは "index.html.eex" というテンプレートを用いて HTML 文書を生成し、ブラウザに返します。

ここで定義した関数 index/2 の第一引数 conn は「Plug.Conn 構造体」と呼ばれます。『初級①』ではその名前だけを紹介し、解説は後回しにしました。「構造体」という概念については次章（第 4 章）で解説します。とりあえず、Phoenix のアクションは Plug.Conn 構造体を受け取って、関数 render/2 の第一引数に指定する、という点を記憶にとどめて先に進みましょう。

> Ruby on Rails にはコントローラおよび関連ファイルのひな型を生成するジェネレータが存在しますが、改訂時点（2017 年 11 月 26 日）における Phoenix の最新版（1.2.5）には存在しません。名前から類推すると mix phoenix.gen.html というコマンドでできそうですが、これは Rails の rails g scaffold に相当するコマンドです。

ビューモジュールの作成

第 3 の手順は、top コントローラ用のビューモジュールの作成です。web/views ディレクトリに新規ファイル top_view.ex を次の内容で作成してください。

web/views/top_view.ex (New)

```
1  defmodule NanoPlanner.TopView do
2    use NanoPlanner.Web, :view
3  end
```

ビューモジュールの役割については『初級①』第 15 章で解説しました。忘れてしまった方は復習してください。

> Phoenix のビューモジュールは、Ruby on Rails のヘルパーモジュールにほぼ相当します。テンプレートで使用するためのヘルパー関数（Rails 用語ではヘルパーメソッド）を定義するためのモジュールです。ただし、Rails のヘルパーメソッド群が全アプリケーションで共有されるのに対し、Phoenix のヘルパー関数群はコントローラ単位で分離されている、という違いがあります。

テンプレートの作成

第 4 の手順（最後の手順）は、top#index アクションのための EEx テンプレートの作成です。まず、テンプレートを置くためのディレクトリを作成してください。

```
$ mkdir -p web/templates/top
```

次に、このディレクトリに新規ファイル index.html.eex を次の内容で作成してください。

web/templates/top/index.html.eex (New)

```
1  <p>top#index</p>
```

これでトップページを表示できるようになりました。mix phoenix.server コマンドで Phoenix サーバーを起動し、ブラウザで http://localhost:4000 を開くと、図 3.1 のような画面が表示されます。

図 3.1　トップページ（初期状態）

ターミナルに戻って Ctrl-C を 2 回入力し、Phoenix サーバーを止めてください。

3.2 フロントエンド開発の基盤を整える

本節では、ビジュアルデザイン面での開発を効率化するため、フロントエンド開発用のライブラリを NanoPlanner に導入していきます。『初級①』の第 10 章と第 11 章で行ったことの繰り返しですので、細かい解説は省略して作業手順のみを示します。

sass-brunch の導入

ターミナルで、次のコマンドを実行してください。

```
$ npm install --save-dev sass-brunch
```

web/static/css ディレクトリの app.css を削除し、同ディレクトリに空の新規ファイル app.scss（拡張子に注意）を作成します。

```
$ rm web/static/css/app.css
$ touch web/static/css/app.scss
```

Bootstrap の導入

ターミナルで次のコマンドを実行してください。

```
$ npm install --save jquery popper.js tether bootstrap@4.0.0-beta.2
```

> 『Elixir/Phoenix 初級①』（初版）第 11 章では、ここで web/static/css/app.scss に Bootstrap の CSS を読み込む @import ディレクティブを追加しました。しかし、出版後にこの手順は不要であることが判明しましたので、本巻ではこの手順を省略しています。

テキストエディタで brunch-config.js を次のように編集してください。

brunch-config.js

```
  :
66      npm: {
67 -      enabled: true
67 +      enabled: true,
68 +      styles: {
69 +        bootstrap: ["dist/css/bootstrap.css"]
70 +      },
71 +      globals: {
72 +        $: "jquery",
73 +        jQuery: "jquery",
74 +        Popper: "popper.js",
75 +        Tether: "tether"
76 +      }
77 +    },
78 +
79 +    watcher: {
80 +      usePolling: true
81      }
82  };
```

watcher.usePolling オプションに true をセットすると、Brunch によるファイルの変更の検知がほんの少しだけ遅くなりますが、動作が安定します。

web/static/js/app.js を次のように編集してください。

web/static/js/app.js

```
  :
10  // Import dependencies
11  //
12  // If you no longer want to use a dependency, remember
13  // to also remove its path from "config.paths.watched".
14  import "phoenix_html"
15 + import "bootstrap"
16
17  // Import local files
```

■コラム: JavaScript コードの置き場所

ブラウザが Web アプリケーションから返ってきた HTML 文書を読み込んだときに、JavaScript コードを実行したい場合があります。例えば、Bootstrap の Tooltip コンポーネントを有効にするには、つぎのようなコードを走らせる必要があります。

```
$(function() {
  $('[data-toggle="tooltip"]').tooltip();
});
```

このコードはどこに設置すればいいでしょうか。tooltip.js のような名前のファイルとして web/static/js ディレクトリに置けばよさそうに思えますが、実は web/static/vendor ディレクトリに置くのが正解です。

web/static/js ディレクトリに置かれる JavaScript ファイルは、ES6 モジュールの形式で記述する必要があります（本巻では扱いません）。その形式で書かれていない JavaScript ファイルは web/static/vendor ディレクトリに置いてください。

Font Awesome

ターミナルで以下のコマンドを順に実行してください。

```
$ npm install --save font-awesome
$ npm install --save-dev copycat-brunch
```

web/static/css ディレクトリの app.scss（現時点では空）に次の内容を書き込みます。

web/static/css/app.scss

```
1 + @import "node_modules/font-awesome/scss/font-awesome";
```

アプリケーションルートディレクトリにある brunch-config.js を次のように書き換えてください。

brunch-config.js

```
  :
52    // Configure your plugins
53    plugins: {
54      babel: {
55        // Do not use ES6 compiler in vendor code
56        ignore: [/web\/static\/vendor/]
57 -    }
57 +    },
58 +    copycat: {
59 +      fonts: ["node_modules/font-awesome/fonts"]
60 +    }
61    },
  :
```

ファビコン

ターミナルで以下のコマンドを順に実行してください。

```
$ pushd web/static/assets/images
$ wget https://www.oiax.jp/books/files/pico_planner.ico
$ mv pico_planner.ico nano_planner.ico
$ wget https://www.oiax.jp/books/files/clock256.png
$ popd
```

pushd コマンドはカレントディレクトリを「スタック」という記憶領域に保存してから、指定されたディレクトリに移動します。「スタック」に保存されたディレクトリに戻るには *popd* コマンドを使用します。*popd* コマンドによって、最後に保存されたディレクトリがスタックから取り除かれます。

ファビコンをレイアウトテンプレートに埋め込みます。

web/templates/layout/app.html.eex

```
  :
10      <title>NanoPlanner</title>
11      <%= tag :link, rel: "stylesheet",
12        href: static_path(@conn, "/css/app.css") %>
13 +    <%= tag :link, rel: "shortcut icon",
14 +      href: static_path(@conn, "/images/nano_planner.ico") %>
15 +    <%= tag :link, rel: "apple-touch-icon",
16 +      href: static_path(@conn, "/images/clock256.png") %>
17    </head>
  :
```

mix phoenix.server コマンドで Phoenix サーバーを起動し、ブラウザで http://localhost:4000 を開くと、タブにファビコンが表示されます（図 3.2）。

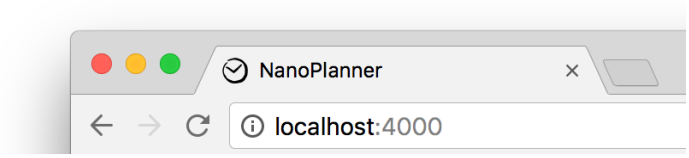

図 3.2　タブにファビコンを表示

ブラウザはファビコンのデータをキャッシュに保存するため、ファビコンファイルを置き換えても表示が変わらないことがあります。その場合は、ブラウザ自体を完全に終了して起動しなおしてください。

3.3　body 要素の分割

body 要素の内側を、ヘッダ部、メイン部、フッタ部に三分割し、ヘッダ部にトップページへのリンクを設置します。『初級①』の第 10 章で行ったことの繰り返しですので、作業手順のみを示します。

ヘッダ部、メイン部、フッタ部

web/templates/layout ディレクトリのファイル app.html.eex（レイアウトテンプレート）を次のように書き換えてください。

web/templates/layout/app.html.eex

```
:
19      <body>
20 +      <header>
21 +        <h1><%= link "NanoPlanner", to: top_path(@conn, :index, []) %></h1>
22 +      </header>
23 +      <main>
:
```

さらに、同ファイルを次のように書き換えます。

web/templates/layout/app.html.eex

```
:
24 -      <%= render @view_module, @view_template, assigns %>
24 +        <%= render @view_module, @view_template, assigns %>
25 +      </main>
26 +      <footer>
27 +        &copy; 2017 Oiax Inc.
28 +      </footer>
29        <%= content_tag :script, "", src: static_path(@conn, "/js/app.js") %>
:
```

スタイルシートの適用

メイン部に対するスタイルシートを新規作成します。

web/static/css/main.scss (New)

```
1   main {
2     margin: 1rem;
3   }
```

続いて、ヘッダ部に対するスタイルシートを新規作成します。

web/static/css/header.scss (New)

```
1  header {
2    background-color: #ddd;
3    border-style: solid;
4    border-width: 1px;
5    border-color: #ccc;
6
7    h1 {
8      font-size: 1.25rem;
9      margin: 0.5rem;
10   }
11 }
```

同様に、フッタ部に対するスタイルシートを新規作成します。

web/static/css/footer.scss (New)

```
1  footer {
2    background-color: #eee;
3    color: #777;
4    padding: 0.5rem;
5    font-size: 0.75rem;
6    font-family: Helvetica, Arial, sans-serif;
7  }
```

ブラウザの画面は図 3.3 のように変化します。

図 3.3　トップページ（スタイルシートの適用後）

『初級①』第 8 章で説明したライブリロード機能のおかげで、テンプレートとビューのソースファイル、アセット（JavaScript、スタイルシート、画像）のファイル、翻訳データなどが書き換わったときには、自動でブラウザが再読み込みされます。なお、ライブリロード機能が有効になるのは Phoenix サーバーが dev モードで動作している場合のみです。また、web/router.ex やコントローラのソースファイルは監視の対象とならないので、手動で再読み込みする必要があります。

ターミナルに戻って Ctrl-C を 2 回入力し、Phoenix サーバーを停止してください。