第 2 章

開発プロジェクト始動！

NanoPlanner 開発プロジェクトの始まりです。まず、Elixir/Phoenix の開発環境が整っていることを確認してから、ソースコードの骨格を作ります。そして、Phoenix サーバーを起動・終了させる練習をしてから、ソースコードの整理・整頓を行います。

2.1 各種ソフトウェアのインストール状況の確認

はじめに、Elixir/Phoenix の開発環境が整っているかどうかを確認しましょう。もし整っていない場合は、付録 A を参照して各種ソフトウェアをインストールしてください。

まず、次のコマンドを実行してください。

```
$ mix hex.info
```

ここで `bash: mix: command not found` のようなメッセージが出力される場合、Elixir がインストールされていません。

インストールされていれば、次のような結果が出力されます。

```
Hex:    0.18.2
Elixir: 1.7.4
OTP:    21.1

Built with: Elixir 1.7.4 and OTP 19.3
```

2 行目と 3 行目のバージョン番号を確認してください。これらの番号は大きす

ぎても小さすぎても支障が出る可能性があります。

次に、Phoenix のバージョン番号を調べます。

```
$ mix phoenix.new -v
```

ここで `** (Mix) The task "phoenix.new" could not be found` のようなメッセージが出力される場合、Phoenx がインストールされていません。インストールされていれば、`Phoenix v1.2.4` のようにバージョン番号が出力されます。本書の内容は Phoenix 1.2.4 に基づいてますので、`v1.2.4` とは異なる番号が出力される場合は、Phoenix の再インストールが必要です。

> 【重要な注意】 Phoenix 1.2 系の最新バージョンは `1.2.5` ですが、筆者の調べた限り、2018 年 12 月 2 日現在の環境ではうまく動かすことができません。必ずバージョン `1.2.4` を利用してください。

続いて、Node.js（サーバーサイド JavaScript 環境）のバージョン番号を調べます。

```
$ node --version
```

`v10.14.1` のように出力されれば OK です。

2.2 データベース管理システム（DBMS）の選択

NanoPlanner の開発に着手しましょう。前巻で作った ModestGreeter とは異なり、今回はデータベースを利用しますので、最初の作業は**データベース管理システム**（DBMS）を選ぶことです。

データベースとは情報の集合です。情報の単なる寄せ集めではなく、検索や集計がすばやくできるように構造化されたものです。このデータベースを操作するソフトウェアがデータベース管理システムです。

Phoenix が対応しているのは、次の 4 種類です。

- PostgreSQL

- MySQL
- Microsoft SQL Server
- MongoDB

> Phoenix 1.1 までは SQLite3 にも対応していましたが、Phoenix 1.2 では使用できません。アダプタ `sqlite_ecto` が Ecto 2（Phoenx 1.2 がデータベース操作に使用するライブラリ）に対応していないためです。将来的には、SQLite3 サポートが復活する可能性があります。

本書では、PostgreSQL または MySQL を採用します。PostgreSQL をインストールする手順については付録 B を参照してください。MySQL をインストールする手順については付録 C を参照してください。

2.3 ラクダ、ヘビ、鎖

次に進む前に、アルファベットの表記法について説明しておきます。

私たちがこの巻で作成する Web アプリの名前は NanoPlanner です。この名前は“nano”と“planner”というふたつの単語から構成されるわけですが、単語間を空白文字で連結するのではなく、それぞれの単語の先頭を大文字に変えて直接連結しています。この表記法は**キャメルケース**（camel case）と呼ばれます。“camel”は「ラクダ」を意味する英単語です。大文字の部分が「ラクダのこぶ」に見えることに由来します。

“nano”と“planner”からひと続きの文字列を作る方法は他にもあります。ひとつは空白文字の代わりにアンダースコア（_）を用いて“nano_planner”のように連結するものです。この表記法を**スネークケース**（snake case）と呼びます。“snake”は爬虫類のヘビを意味します。

もうひとつが、マイナス記号（-）で単語間を連結する**チェーンケース**（chain case）という表記法です。“chain”は鎖という意味ですね。“nano”と“planner”から“nano-planner”という文字列が作られます。チェーンケースは HTML の `id` 属性や `class` 属性の値としてしばしば使われます。

2.4 Welcome ページの表示

ソースコードの骨格の生成

では、NanoPlanner のソースコードの骨格（skeleton）を作成しましょう。以下のコマンドを順に実行してください。

```
$ mkdir -p ~/projects
$ cd ~/projects
$ mix phoenix.new nano_planner --database postgres
```

> DBMS として MySQL を利用する方は、postgres を mysql で置き換えてください。

ここでターミナルに次のようなメッセージが表示されるので、Enter キーを押してください。

```
Fetch and install dependencies? [Yn]
```

新たに nano_planner というディレクトリができています。*cd* コマンドでそこに移動してください。

```
$ cd nano_planner
```

mix phoenix.new コマンドの使い方については『初級①』第 5 章で学習しました。第 1 引数にはソースコードを格納するディレクトリを指定します。通常、ディレクトリ名は全部小文字で表記するか、スネークケースで表記します。

『初級①』ではデータベースアクセス用のパッケージ Ecto を使用しないので --no-ecto オプションを指定しましたが、今回は指定しません。その代わり、--database オプションを用いて DBMS の種類を指定しています。このオプションに指定できるのは、postgres、mysql、mssql、mongodb のいずれかです。デフォルトは postgres です。

■コラム: ベースモジュール

mix phoenix.new コマンドの第 1 引数に指定された文字列（ディレクトリ目）は、キャメルケースに変換された後、ソースコードの中で**ベースモジュール**の名前として使われます。

ベースモジュールは Phoenix アプリケーションのソースコードで随所に現れます。例えば、web/router.ex の 1 行目をご覧ください。

```
defmodule NanoPlanner.Router do
```

ここに現れる NanoPlanner がベースモジュールです。

mix phoenix.new コマンドに指定したディレクトリ名から作られる名前とは別の名前のベースモジュールを使いたい場合は、次のように mix phoenix.new コマンドに --module オプションを加えます。

```
$ mix phoenix.new temp/np --module NanoPlanner
```

このコマンドを使用した場合、NanoPlanner をベースモジュール名とする Phoenix アプリケーションの骨格が temp/np ディレクトリに生成されます。

依存パッケージの調整

mix.exs を次のように修正します。

mix.exs

```
  :
32    defp deps do
33      [{:phoenix, "~> 1.2.4"},
34       {:phoenix_pubsub, "~> 1.0"},
35 -     {:phoenix_ecto, "~> 3.0"},
35 +     {:phoenix_ecto, "~> 3.4.0"},
36       {:postgrex, ">= 0.0.0"},
37       {:phoenix_html, "~> 2.6"},
```

```
38        {:phoenix_live_reload, "~> 1.0", only: :dev},
39        {:gettext, "~> 0.11"},
40 -      {:cowboy, "~> 1.0"}]
40 +      {:cowboy, "~> 1.0"},
41 +      {:plug_cowboy, "~> 1.0"}]
42      end
 :
```

ターミナルで次のコマンドを実行してください。

```
$ mix deps.update --all
```

2018 年 12 月 2 日現在、mix.exs を修正せずに mix deps.get を実行すると phoenix_ecto パッケージのバージョン 3.5.0 がインストールされます。しかし、このバージョンは正常に動作しません。そこでバージョン 3.4.0 にダウングレードしています。また、plug_cowboy（Erlang 製 Web サーバーである Cowboy のためのアダプター）を追加しています。Phoenix 1.2.4 がリリースされた当時（2017 年 5 月）にはこのパッケージは不要でしたが、現在の環境ではこれがないと Phoenix アプリケーションを起動することができません。

mix.exs の修正

mix.exs を次のように修正します。

mix.exs

```
 :
50      defp aliases do
51        ["ecto.setup": ["ecto.create", "ecto.migrate", "run priv/repo/seeds.exs"],
52          "ecto.reset": ["ecto.drop", "ecto.setup"],
53 -        "test": ["ecto.create --quiet", "ecto.migrate", "test"]]
53 +        test: ["ecto.create --quiet", "ecto.migrate", "test"]]
54      end
55    end
```

この修正をしないと *mix* コマンドを実行するたびに、次のような警告メッセー

ジがターミナル上に出力されます。

```
warning: found quoted keyword "test" but the quotes are not required. Note >
that keywords are always atoms, even when quoted, and quotes should >
only be used to introduce keywords with foreign characters in them
  mix.exs:53
```

キーワードリスト（『初級①』第 14 章）のキーは常にアトムであるので、特殊記号を含まない限り引用符で囲む必要はないという意味の警告です。あくまで警告であるため無視してもいいのですが、*mix* コマンドを実行する頻度はとても高いので、警告の原因を除去しておきます。

データベース接続設定の確認と変更

続いて、Phoenix アプリケーションがデータベースに接続するための設定情報を確認しましょう。エディタで config ディレクトリの下にある dev.exs を開いてください。

> このファイルに書かれているのは、開発モード（dev 環境）における設定情報です。テストモード（test 環境）および本番モード（prod 環境）向けの設定情報は、同じディレクトリにある test.exs および prod.exs に記載されています。

PostgreSQL を利用する場合

付録 B に従って PostgreSQL をセットアップした場合、phoenix というパスワードを持つ phoenix ユーザーが作成されています。そこで、dev.exs の末尾を次のように書き換えてください。

config/dev.exs

```
37    config :nano_planner, NanoPlanner.Repo,
38      adapter: Ecto.Adapters.Postgres,
39 -    username: "postgres",
39 +    username: "phoenix",
40 -    password: "postgres",
40 +    password: "phoenix",
```

```
41    database: "nano_planner_dev",
42    hostname: "localhost",
43    pool_size: 10
```

必要に応じて、ユーザー名、パスワード、データベース名、ホスト名を適宜変更してください。
デフォルトの 5432 番以外のポートを利用する場合は port オプションをセットしてください。

MySQL を利用する場合

付録 C に従って MySQL をセットアップした場合、phoenix というパスワードを持つ phoenix ユーザーが作成されています。そこで、dev.exs の末尾を次のように書き換えてください。

config/dev.exs

```
37    config :nano_planner, NanoPlanner.Repo,
38      adapter: Ecto.Adapters.MySQL,
39 -    username: "root",
39 +    username: "phoenix",
40 -    password: "",
40 +    password: "phoenix",
41      database: "nano_planner_dev",
42      hostname: "localhost",
43      pool_size: 10
```

必要に応じて、ユーザー名、パスワード、データベース名、ホスト名を適宜変更してください。
デフォルトの 3306 番以外のポートを利用する場合は port オプションをセットしてください。

データベースの作成

ターミナルで次のコマンドを実行します。

```
$ mix ecto.create
```

ターミナルにさまざまなコンパイルログが出力された後に、次のようなメッセージが出力されれば成功です。

```
The database for NanoPlanner.Repo has been created
```

Phoenix サーバーの起動と停止

ターミナルで次のコマンドを実行します。

```
$ mix phoenix.server
```

ターミナルに次のように出力されれば、Phoenix サーバーが起動しています。

```
[info] Running NanoPlanner.Endpoint with Cowboy using http://localhost:4000
00:02:57 - info: compiled 6 files into 2 files, copied 3 in 1.1 sec
```

ブラウザで `http://localhost:4000` を開くと、図 2.1 のような画面が表示されます。

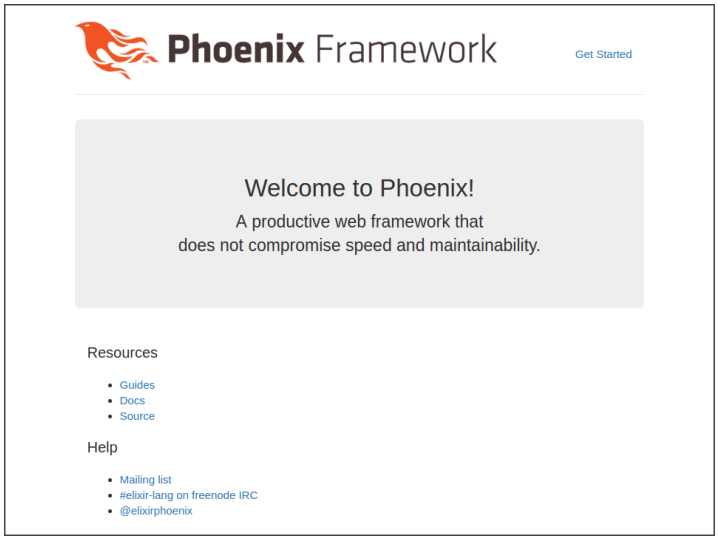

図 2.1　Welcome to Phoenix!

ターミナルに戻って `Ctrl-C` を 2 回続けて入力してください。Phoenix サーバーが停止します。

2.5　下ごしらえ

`mix phoenix.new` コマンドにより生成された Phoenix アプリケーションの骨格には、「Welcome to Phoenix!」ページを表示するためのコードが含まれています。今後の開発にとっては余分なので、削除してしまいましょう。

ここから章末までは、『初級①』第 5 章の後半と内容が重複しています。ただし、まったく同じではありません。『初級①』第 17 章でレイアウトに対して行う変更を前もって取り込んでいます。

不要なファイル、ディレクトリの削除

まず、以下のコマンドを順に実行し、不要なファイルとディレクトリを削除します。

```
$ rm web/controllers/page_controller.ex
$ rm web/static/css/phoenix.css
$ rm web/views/page_view.ex
$ rm -rf web/templates/page
$ rm test/controllers/page_controller_test.exs
```

web/router.ex の整理

続いて、web/router.ex のソースコードの一部を除去します。

web/router.ex

```
 1  defmodule NanoPlanner.Router do
 2    use NanoPlanner.Web, :router
 3
 4    pipeline :browser do
 5      plug :accepts, ["html"]
 6      plug :fetch_session
 7      plug :fetch_flash
 8      plug :protect_from_forgery
 9      plug :put_secure_browser_headers
10    end
11 -
12 -  pipeline :api do
13 -    plug :accepts, ["json"]
14 -  end
11
12    scope "/", NanoPlanner do
 :
```

11〜14 行のソースコードは Phoenix で API サーバーを作るためのものです。NanoPlanner では使用しません。

さらに、同ファイルを次のように書き換えます。

web/router.ex

```
 :
 9       plug :put_secure_browser_headers
10     end
11
12     scope "/", NanoPlanner do
13 -     pipe_through :browser # Use the default browser stack
13 +     pipe_through :browser
14 -
15 -     get "/", PageController, :index
14     end
 :
```

13 行目の行末のコメントを除去しました。15 行目は、「Welcome to Phoenix!」ページを表示するためのものですので、14 行目の空行とともに削除します。

> 13 行目で使われている pipe_through/1 マクロは、HTTP リクエストに対してどのような前処理を行うかを指定します。引数としてアトム :atom を指定した場合、ブラウザからのリクエストに応えるために必要な前処理、例えばセッション処理や CSRF 対策などを行います。通常の Web サイトを構築するために Phoenix を利用する場合、この行はそのままにしておく必要があります。

最後に、ソースコード末尾近くのコメントを削除します。

web/router.ex

```
 :
14     end
15 -
16 -   # Other scopes may use custom stacks.
17 -   # scope "/api", NanoPlanner do
18 -   #   pipe_through :api
19 -   # end
15   end
```

レイアウトテンプレートの整理

次に、web/templates/layout ディレクトリにある app.html.eex ファイルを書き換えます。

web/templates/layout/app.html.eex

```
  :
10 -     <title>Hello NanoPlanner!</title>
10 +     <title>NanoPlanner</title>
11 -     <link rel="stylesheet" href="<%= static_path(@conn, "/css/app.css") %>">
11 +     <%= tag :link, rel: "stylesheet",
12 +       href: static_path(@conn, "/css/app.css") %>
13     </head>
  :
```

title 要素の中身を変更しました。また、link 要素を tag 関数を使って生成するように書き換えました。

> 11 行目を書き換えた理由については『初級①』第 17 章で説明しました。筆者は、HTML タグの内側に <%= %> で値が埋め込まれるのが好きではないのでこうしています。

同ファイルの 16〜29 行を削除します。

web/templates/layout/app.html.eex

```
  :
15     <body>
16 -     <div class="container">
17 -       <header class="header">
  :
29 -       <main role="main">
16           <%= render @view_module, @view_template, assigns %>
17         </main>
  :
```

さらに、同ファイルを書き換えます。

web/templates/layout/app.html.eex

```
 :
16 -         <%= render @view_module, @view_template, assigns %>
16 +     <%= render @view_module, @view_template, assigns %>
17 -       </main>
18 -
19 -     </div> <!-- /container -->
17       <script src="<%= static_path(@conn, "/js/app.js") %>"></script>
18     </body>
19 </html>
```

さらに、同ファイルを書き換えます。

web/templates/layout/app.html.eex

```
 :
16       <%= render @view_module, @view_template, assigns %>
17 -     <script src="<%= static_path(@conn, "/js/app.js") %>"></script>
17 +     <%= content_tag :script, "", src: static_path(@conn, "/js/app.js") %>
18     </body>
19 </html>
```

script 要素を content_tag 関数を使って生成するように書き換えました。11 行目の link 要素を tag 関数を使って書き換えたのと同じ理由です。

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 brunch@2.10.17
```

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@4.1.3
```

> 『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@4.7.0
$ 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）。

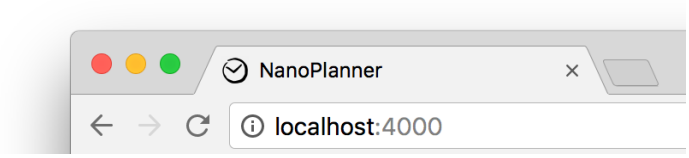

図 3.2　タブにファビコンを表示

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。