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、『Elixir/Phoenix 初級①』（第 2 版）の第 2 章、付録 C、付録 D 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。

『Elixir/Phoenix 初級①』（第 2 版）の第 2 章、付録 C、付録 D については、本書の読者サポートページ（http://www.oiax.jp/books/elixir-phoenix-volume2.html）で PDF 文書として無料で配布されています。

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:    0.17.1
Elixir: 1.3.4
OTP:    19.3

Built with: Elixir 1.3.4 and OTP 18.3
```

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ます。

> Elixir 1.3 系は Erlang/OTP 20 系と相性がよくありません。2017 年 11 月 20 日現在、Homebrew や APT で Erlang/OTP を普通にインストールするとバージョン 20.1 がインストールされてしまいます。ダウングレードが必要です。

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 v1.2.5` 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 1.2.5 に基づいてますので、`v1.2.5` 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。

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
```

`v8.9.1` のように出力されれば OK です。ただし、3 番目のマイナーバージョン番号は異なるかもしれません。`v8.9.0` より小さな番号が出力される場合は、Node.js のアップデートが必要です。以下のコマンドを順に実行してください。

```
$ nvm install 8.9.1
$ nvm alias default 8.9.1
```

nvm コマンドが見つからない場合は、『初級①』付録 D を参照してインストー

ルしてください。

```
http://www.oiax.jp/books/files/ex01-chxd.pdf
```

改めて Node.js のバージョン番号を調べます。

```
$ node --version
```

ターミナルに、v8.9.1 と表示されることを確認してください。
さらに、*npm*（Node.js 用のパッケージマネージャ）を更新します。

```
$ npm update -g npm
```

npm のバージョン番号を確認します。

```
$ npm --version
```

ターミナルに 5.5.1 よりも大きなバージョン番号が出力されれば 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 A 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 B 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。

Ecto のダウングレード

mix.exs を次のように修正します。

mix.exs

```
1     defp deps do
2       [{:phoenix, "~> 1.2.5"},
3        {:phoenix_pubsub, "~> 1.0"},
4        {:phoenix_ecto, "~> 3.0"},
5        {:postgrex, ">= 0.0.0"},
6        {:phoenix_html, "~> 2.6"},
7        {:phoenix_live_reload, "~> 1.0", only: :dev},
8 +      {:ecto, "~> 2.1.6"},
9        {:gettext, "~> 0.11"},
10       {:cowboy, "~> 1.0"}]
11    end
```

ターミナルで次のコマンドを実行してください。

```
$ mix deps.get
```

2017 年 11 月 26 日現在、デフォルトでインストールされる Ecto（データベースアクセス用のパッケージ）のバージョンは 2.2.6 です。しかし、バージョン 2.2 以降の Ecto は、Elixir 1.3/Phoenix 1.2 と相性がよくありません。バージョン 2.1 系にダウングレードしてください。

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 A 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 B 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
```

ターミナルに次のように出力されれば成功です。

```
==> connection
Compiling 1 file (.ex)
Generated connection app
…
Compiling 12 files (.ex)
Generated nano_planner app
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。

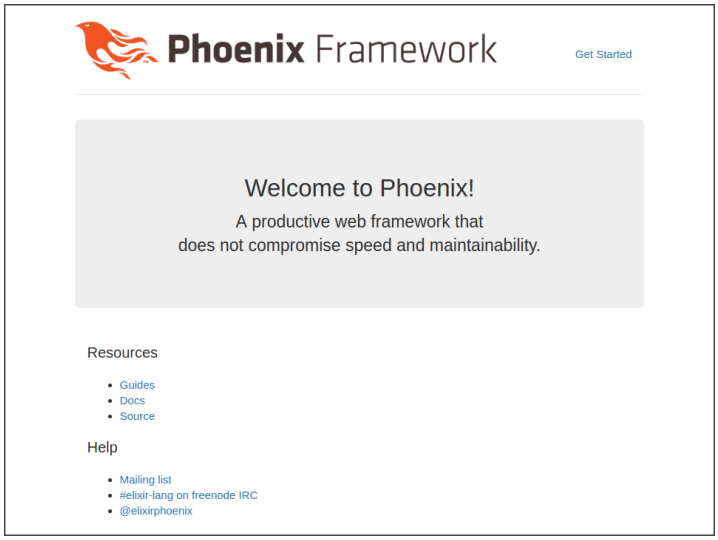

図 2.1　Welcome to Phoenix!

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。