第 6 章

今日の予定表 (1)

この章では、「今日の予定表」を表示する of_today アクションを作りながら、日付時刻で検索対象を絞り込む方法を学びます。ちょっと難しいところもありますが、がんばってください。

6.1 フッタ部の変更

まず初めに、簡単な修正から始めましょう。フッタ部分の Copyright 表示に『初級③』の原稿が書かれた「2017」という年が表示されています。ここに現在の西暦年が表示されるようにしましょう。

テキストエディタで lib/nano_planner_web/views ディレクトリにある layout_view.ex を開いて、次のように関数 this_year/0 を定義してください。

lib/nano_planner_web/views/layout_view.ex

```
1   defmodule NanoPlannerWeb.LayoutView do
2     use NanoPlannerWeb, :view
3 +
4 +   def this_year do
5 +     time_zone = Application.get_env(:nano_planner, :default_time_zone)
6 +     Timex.now(time_zone).year
7 +   end
8   end
```

さらに、lib/nano_planner_web/templates/layout ディレクトリにある

app.html.eex を開いて、28 行目を次のように書き換えてください。

lib/nano_planner_web/templates/layout/app.html.eex

```
 :
27      <footer>
28 -      &copy; 2017 Oiax Inc.
28 +      &copy; <%= this_year() %> Oiax Inc.
29      </footer>
30      <%= content_tag :script, "", src: static_path(@conn, "/js/app.js") %>
31    </body>
32  </html>
```

今年が 2018 年であれば、ブラウザを再読み込みするとフッタ部分が図 6.1 のように変化します。

図 6.1　フッタに現在の西暦年を表示

6.2　七つの基本アクション（前巻の復習）

さて、前巻で私たちは「七つの基本アクション」という概念を学びました（表 6.1）。Phoenix ではデータベーステーブルを扱う典型的な機能群を作るとき、通常これらのアクションを順に実装していくことになります。

Phoenix 用語の「アクション」とは、コントローラモジュールのパブリックな関数のことです。アクションには「動詞」と「URL パターン」が結び付けられます。

表 6.1　七つの基本アクション

アクション名	動詞	URL パターンの例	主な役割
index	GET	/plan_items	リスト表示
new	GET	/plan_items/new	新規追加フォームの表示
create	POST	/plan_items	新規追加
show	GET	/plan_items/:id	詳細表示
edit	GET	/plan_items/:id/edit	変更フォームの表示
update	PUT	/plan_items/:id	変更
delete	DELETE	/plan_items/:id	削除

動詞は「HTTP 動詞」あるいは「HTTP メソッド」とも呼ばれます。ブラウザが Web サーバーに対して送るリクエストの種類を示す文字列です。動詞として使える文字列には GET、POST、PUT、DELETE などがあります。動詞 GET は、ブラウザが Web サーバーから情報を得るときに使用します。動詞 POST は Web サーバーに新しい情報を追加するときに、動詞 PUT は Web サーバーにある既存の情報を書き換えるときに、動詞 DELETE は Web サーバーにある既存の情報を削除するときに使用します。

アクションに結び付けられた動詞と URL パターンの組み合わせを「リソース」と呼びます。リソースは 3 種類に分類されます。集合を扱う「コレクションリソース」、集合の要素を扱う「メンバーリソース」、新規追加フォームための特別な「ニューリソース」です。七つの基本アクションの中では、index アクションと create アクションのためのリソースが「コレクションリソース」に分類されます。そして、new アクションのためのリソースが「ニューリソース」に分類され、残りの 4 個のアクションのためのリソースが「メンバーリソース」に分類されます。

リソースの種類を見分けるのは簡単です。URL パターンに :id を含めばメンバーリソースです。そうでなければコレクションリソースかニューリソースですが、ニューリソースは「新規追加フォームの表示」という特殊な用途のリソースですので、たいていはコレクションリソースです。

最後に、「経路（route）」という言葉を思い出してください。アクションとリソースの組み合わせを指す Phoenix 用語です。経路を設定するファイルが lib/nano_planner_web ディレクトリにある router.ex です。Phoenix アプリ

の最も重要なファイルです。

6.3 of_today アクション

経路設定

前章（第 1 章）で紹介した新機能群のうち「今日の予定表」を表示する機能を作り始めることにしましょう。この機能にリソースを割り当てるとすれば、3 種類のうちどれが適当でしょうか。そして、動詞は何を使うべきでしょうか。

「今日の予定表」は複数の予定項目に関する情報を表示する機能ですので、集合を扱うコレクションリソースと考えるべきです。この機能では情報の追加、変更、削除は行いません。単に情報を取得するだけです。したがって、動詞は GET です。

アクション名については特に決まりはありませんが、筆者はコレクションリソースのためのアクションには、英語の形容詞またはそれに準ずる語句（動詞の過去分詞や前置詞句）を採用することにしています。例えば、active、deleted、under_review、などです。ここでは「今日の」を意味する of_today をアクション名として採用することにしましょう。

最後に URL パターンを決めましょう。筆者は /plan_items/of_today がいいと思います。

では、このリソースへの経路を登録しましょう。登録先は router.ex です。現在、次のような内容を持っています。

lib/nano_planner_web/router.ex

```
 :
12    scope "/", NanoPlannerWeb do
13      pipe_through :browser
14
15      get "/", TopController, :index
16      get "/lessons/form", LessonController, :form
17      get "/lessons/register", LessonController, :register
18      get "/lessons/hello", LessonController, :hello
19
20      resources "/plan_items", PlanItemController
21    end
```

```
22  end
```

マクロ resources/3 については『初級③』の第 6 章で学びました。これを利用すると七つの基本アクションに対する経路を一度に設定できます。20 行目は次のように書いても構いません。

```
get "/plan_items", PlanItemController, :index
get "/plan_items/new", PlanItemController, :new
post "/plan_items", PlanItemController, :create
get "/plan_items/:id", PlanItemController, :show
get "/plan_items/:id/edit", PlanItemController, :edit
put "/plan_items/:id", PlanItemController, :update
patch "/plan_items/:id", PlanItemController, :update
delete "/plan_items/:id", PlanItemController, :delete
```

でも、こんな風に長々と書くよりも、マクロ resources/3 を使って 1 行で書くほうが簡潔で分かりやすいですね。

7 番目の記述は動詞 PATCH と URL パターン plan_items/:id の組み合わせをリソースとして登録しています。update アクションは動詞 PUT でも動詞 PATCH でも呼び出せます。

七つの基本アクション以外のアクションへの経路設定

では、実際に「今日の予定表」機能にリソースを割り当てて経路を設定しましょう。router.ex を次のように書き換えてください。

lib/nano_planner_web/router.ex

```
  :
  :
12    scope "/", NanoPlannerWeb do
13      pipe_through(:browser)
14
15      get "/", TopController, :index
16      get "/lessons/form", LessonController, :form
17      get "/lessons/register", LessonController, :register
```

```
18      get "/lessons/hello", LessonController, :hello
19 +
20 +    scope "/plan_items" do
21 +      get "/of_today", PlanItemController, :of_today
22 +    end
23
24      resources "/plan_items", PlanItemController
25    end
26  end
```

マクロ scope/3 を用いてネストされた経路を追加しました。実は、20〜22 行目は次のように 1 行で書くことも可能です。

```
get "/plan_items/of_today", PlanItemController, :of_today
```

しかし、今後も /plan_items で始まる URL パターンに経路を設定する機会が出てくると考えられるし、マクロ resources/3 で定義される経路群の仲間であることを明示する意味もあるので、マクロ scope/3 を使って記述しました。

> Ruby on Rails の経路設定では、resources メソッドにブロック（do ... end で囲まれたコード）を追加し、その内側で get、post、patch、delete などのメソッドを用いてカスタムアクションへの経路設定ができますが、Phoenix ではできません。

トップページにリンクを設置

次に、「今日の予定表」ページへのリンクをトップページに設置しましょう。lib/nano_planner_web/templates/top ディレクトリにある index.html.eex を次のように書き換えてください。

lib/nano_planner_web/templates/top/index.html.eex

```
1  <div class="list-group">
2    <%= link to: Routes.plan_item_path(@conn, :index),
3      class: "list-group-item" do %>
```

```
4       <i class="fa fa-list"></i> 予定表
5     <% end %>
6 +   <%= link to: Routes.plan_item_path(@conn, :of_today),
7 +     class: "list-group-item" do %>
8 +     <i class="fa fa-list"></i> 今日の予定表
9 +   <% end %>
10    <%= link to: Routes.plan_item_path(@conn, :new),
11      class: "list-group-item" do %>
12      <i class="fa fa-plus"></i> 予定の追加
13    <% end %>
14  </div>
```

関数 Routes.plan_item_path/2 は、第 1 引数に Plug.Conn 構造体、第 2 引数にアクションの名前を表すアトムを取ります（『初級①』第 16 章）。

ブラウザでトップページを開き直すと図 6.2 のように表示されます。

図 6.2　トップページに「今日の予定表」リンクが設置された

of_today アクションの仮実装

続いて、of_today アクションに仮の実装を与えます。plan_item コントローラのソースコードを次のように書き換えてください。

lib/nano_planner_web/controllers/plan_item_controller.ex

```
1    defmodule NanoPlannerWeb.PlanItemController do
2      use NanoPlannerWeb, :controller
3      alias NanoPlanner.Schedule
4
5      def index(conn, _params) do
6        plan_items = Schedule.list_plan_items()
7        render(conn, "index.html", plan_items: plan_items)
8      end
9  +
10 +   def of_today(conn, _params) do
11 +     plan_items = Schedule.list_plan_items_of_today()
12 +     render(conn, "index.html", plan_items: plan_items)
13 +   end
14
15     def new(conn, _params) do
 :
```

アクションの中身は index アクションのものとほぼ同じですが、Schedule コンテキストの関数 list_plan_items_of_today/0 を呼ぶように変えてあります。

そして、Schedule コンテキストにその関数を追加します。

lib/nano_planner/schedule/schedule.ex

```
1    defmodule NanoPlanner.Schedule do
2      import Ecto.Query
3      alias NanoPlanner.Repo
4      alias NanoPlanner.Schedule.PlanItem
5
6      def list_plan_items do
7        PlanItem
8        |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
9        |> Repo.all()
10       |> convert_datetime()
11     end
12 +
13 +   def list_plan_items_of_today do
14 +     PlanItem
15 +     |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
```

```
16 +      |> Repo.all()
17 +      |> convert_datetime()
18 +    end
19
20      def get_plan_item!(id) do
 :
```

とりあえず、関数の中身は list_plan_items/0 とまったく同じにしてあります。

では、動作確認をしましょう。Phoenix サーバーを止め、データベースの中身をリセットします。

```
$ mix ecto.reset
```

Phoenix サーバーを起動してから、ブラウザでトップページの「今日の予定表」リンクをクリックすると図 6.3 のように表示されます。

図 6.3　今日の予定表（最初の実装）

この時点では「予定表」のページとまったく同じ結果が表示されます。次の節以降で「今日の予定表」だけが表示されるように作り込んで行きます。

6.4 開始日時による予定項目の絞り込み

マクロ where/3

では、「今日の予定表」だけを表示するように Schedule コンテキストの関数 list_plan_items_of_today/0 を書き換えていきましょう。まず、「今日」という期間の始点と終点を変数 t0 と t1 にセットします。

lib/nano_planner/schedule/schedule.ex

```
  :
 13     def list_plan_items_of_today do
 14 +     t0 = current_time() |> Timex.beginning_of_day()
 15 +     t1 = t0 |> Timex.shift(hours: 24)
 16 +
 17       PlanItem
 18       |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
 19       |> Repo.all()
 20       |> convert_datetime()
 21     end
  :
```

変数 t0 に「今日の午前 0 時ちょうど」を示す時刻、変数 t1 に「明日の午前 0 時ちょうど」を示す時刻をセットしています。関数 current_time/0 は、同じ Schedule モジュールの中で定義されているプライベート関数です。

そして、これらの変数を用いて予定項目を絞り込みます。

lib/nano_planner/schedule/schedule.ex

```
  :
 13     def list_plan_items_of_today do
 14       t0 = current_time() |> Timex.beginning_of_day()
 15       t1 = t0 |> Timex.shift(hours: 24)
 16
 17       PlanItem
 18 +     |> where([i], i.starts_at >= ^t0 and i.starts_at < ^t1)
```

```
19        |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
20        |> Repo.all()
21        |> convert_datetime()
22      end
 :
```

前章で学んだクエリ・バインディングスを利用して絞り込み条件を記述しています。変数 t0 と t1 をクエリ式の中に埋め込むためキャレット記号（^）を前に付けている点に注意してください。

指定された条件は「列 starts_at の値が t0 以降かつ t1 よりも前」という意味になります。この条件だけでは「今日の予定表」を完全に絞り込むことにはなりません。とりあえずの出発点です。

シードデータの変更

開始日時による予定項目の絞り込みがうまく行っているかどうかを確認するため、シードデータ投入スクリプトを書き換えましょう。

priv/repo/seeds.ex

```
 :
21  insert!(%PlanItem{
22    name: "帰省",
23    description: "新幹線の指定席を取る。\nお土産を買う。",
24    starts_at: Timex.shift(time1, years: 1, days: -2),
25    ends_at: Timex.shift(time1, years: 1, days: 3)
26  })
27 +
28 + insert!(%PlanItem{
29 +   name: "DVD鑑賞",
30 +   description: "作品未定",
31 +   starts_at: Timex.shift(time0, hours: 23),
32 +   ends_at: Timex.shift(time0, hours: 25)
33 + })
34 +
35 + insert!(%PlanItem{
36 +   name: "姉の出張",
37 +   description: "札幌",
```

```
38 +    starts_at: Timex.shift(time0, days: -1, hours: 10),
39 +    ends_at: Timex.shift(time0, hours: 17)
40 + })
```

今日の午後 11 時から明日の午前 1 時まで続く予定（「DVD 鑑賞」）と昨日の午前 10 時から今日の午後 5 時まで続く予定（「姉の出張」）を追加しました。現時点での実装では、前者はリストに現れますが、後者は現れないはずです。

Phoenix サーバーを止め、シードデータを投入し直します。

```
$ mix ecto.reset
```

Phoenix サーバーを起動してから、ブラウザで「今日の予定表」のページを開くと、図 6.4 のような画面になります。

図 6.4　今日の予定表（開始日時で絞り込み）

図 6.4 は、今日の日付が 2018 年 9 月 5 日である場合の表示です。読者の皆さんが実際に見るものとは、日付が異なります。

リファクタリング

さて、Schedule モジュールの 2 つの関数 list_plan_items/0 と list_plan_items_of_today/0 の間にはコードの重複があります。次の章へ進む前にリファクタリングをしておきましょう。

まず、プライベート関数 fetch_plan_items/1 を次のように定義します。仮引数 query にはクエリ構造体がセットされます。関数からの戻り値もクエリ構造体です。

lib/nano_planner/schedule/schedule.ex

```
  :
13    def list_plan_items_of_today do
14      t0 = current_time() |> Timex.beginning_of_day()
15      t1 = t0 |> Timex.shift(hours: 24)
16
17      PlanItem
18      |> where([i], i.starts_at >= ^t0 and i.starts_at < ^t1)
19      |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
20      |> Repo.all()
21      |> convert_datetime()
22    end
23 +
24 +  defp fetch_plan_items(query) do
25 +    query
26 +    |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
27 +    |> Repo.all()
28 +    |> convert_datetime()
29 +  end
30
31    def get_plan_item!(id) do
  :
```

19〜21 行のコードと 26〜28 行のコードがまったく同じである点に着目してください。プライベート関数 fetch_plan_items/1 を利用すると、関数 list_plan_items/0 は次のように短く書けます。

lib/nano_planner/schedule/schedule.ex

```
  :
  def list_plan_items do
    PlanItem
-   |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
-   |> Repo.all()
-   |> convert_datetime()
+   |> fetch_plan_items()
  end
  :
```

同様に、関数 `list_plan_items_of_today/0` も次のように短縮できます。

lib/nano_planner/schedule/schedule.ex

```
  :
  def list_plan_items_of_today do
    t0 = current_time() |> Timex.beginning_of_day()
    t1 = t0 |> Timex.shift(hours: 24)

    PlanItem
    |> where([i], i.starts_at >= ^t0 and i.starts_at < ^t1)
-   |> order_by(asc: :starts_at, asc: :ends_at, asc: :id)
-   |> Repo.all()
-   |> convert_datetime()
+   |> fetch_plan_items()
  end
  :
```

第 7 章

今日の予定表 (2)

前章に引き続き、「今日の予定表」を表示する機能を作っていきます。まず、「今日」の範囲内で終了する予定項目が「今日の予定表」に含まれるようにします。その後で、「継続中の予定項目」すなわち「昨日以前に始まって明日以降に終わる予定項目」も今日の予定表に表示されるようにします。

7.1 終了日時による項目の絞り込み

続いて、「今日」の範囲内で終了する予定項目も「今日の予定表」に含まれるように NanoPlanner.Schedule モジュールの関数 list_plan_items_of_today/0 を書き換えましょう。

lib/nano_planner/schedule/schedule.ex

```
  :
11    def list_plan_items_of_today do
12      t0 = current_time() |> Timex.beginning_of_day()
13      t1 = t0 |> Timex.shift(hours: 24)
14
15      PlanItem
16 -    |> where([i], i.starts_at >= ^t0 and i.starts_at < ^t1)
16 +    |> where(
17 +      [i],
18 +      (i.starts_at >= ^t0 and i.starts_at < ^t1) or
19 +        (i.ends_at > ^t0 and i.ends_at <= ^t1)
20 +    )
```

```
21        |> fetch_plan_items()
22      end
 :
```

ここで指定されているのは、次に挙げる 2 つの条件の論理和です。

1. 列 `starts_at` の値が `t0` 以降かつ `t1` よりも前
2. 列 `ends_at` の値が `t0` よりも後かつ `t1` 以前

今日の午前 0 時ちょうどに終わる予定は含まない点と明日の午前 0 時ちょうどに終わる予定を含む点に留意してください。

ブラウザを再読み込みすると、図 7.1 のような画面に変化します。書き換え前にはなかった「姉の出張」という予定（昨日から始まって今日終わる）が出現します。

図 7.1　今日の予定表（終了日時で絞り込み）

7.2 継続中の予定項目リスト

関数 list_continued_plan_items/0 の作成

現時点では、「今日の予定表」には「今日始まる予定項目」と「今日終わる予定項目」しか表示されません。できれば「昨日以前に始まって明日以降に終わる予定項目」も表示されるようにしたいところです。そこで、NanoPlanner.Schedule モジュールに新しく関数 list_continued_plan_items/0 を次のように追加します。

lib/nano_planner/schedule/schedule.ex

```
  :
11    def list_plan_items_of_today do
  :
21      |> fetch_plan_items()
22    end
23 +
24 +  def list_continued_plan_items do
25 +    t0 = current_time() |> Timex.beginning_of_day()
26 +    t1 = t0 |> Timex.shift(hours: 24)
27 +
28 +    PlanItem
29 +    |> where([i], i.starts_at < ^t0 and i.ends_at > ^t1)
30 +    |> fetch_plan_items()
31 +  end
32
33    defp fetch_plan_items(query) do
  :
```

この関数は継続中の予定項目のリストを取得するためのクエリ構造体を返します。開始日時が今日の午前 0 時よりも前であり、かつ終了日時が明日の午前 0 時よりも後であれば「継続中」であると言えます。終了日時が明日の午前 0 時きっかりの予定項目は、普通の「今日の予定項目」の中に含まれるので除外します。

続いて、plan_item コントローラの of_today アクションを次のように書き換えてください。

lib/nano_planner_web/controllers/plan_item_controller.ex

```
 :
10    def of_today(conn, _params) do
11      plan_items = Schedule.list_plan_items_of_today()
12 -    render(conn, "index.html", plan_items: plan_items)
12 +    continued_plan_items = Schedule.list_continued_plan_items()
13 +
14 +    render(
15 +      conn,
16 +      "of_today.html",
17 +      plan_items: plan_items,
18 +      continued_plan_items: continued_plan_items
19 +    )
20    end
21
22    def new(conn, _params) do
 :
```

関数 list_continued_plan_items/0 からの戻り値（スキーマ構造体のリスト）を変数 continued_plan_items にセットし、テンプレートに渡しています。

部分テンプレートの抽出

続いて、この of_today アクションで使用する EEx テンプレートを作成していきますが、その前に index アクション用の EEx テンプレートの一部を部分テンプレートとして抽出します。

まず、index.html.eex の 7 行目から 31 行目までを切り取ります。

lib/nano_planner_web/templates/plan_item/index.html.eex

```
1   <%= for item <- @plan_items do %>
2     <%= render "_delete_confirmation.html", item: item, conn: @conn %>
3   <% end %>
4
5   <div class="container-fluid plan-items">
6     <%= for item <- @plan_items do %>
7 -   <div class="row">
8 -     <div class="col-8 col-md-4">
```

```
 9 -       <span class="plan-item-name">
 :
26 -     <div class="col-12">
27 -       <span class="plan-item-duration">
28 -         <%= format_duration(item) %>
29 -       </span>
30 -     </div>
31 -   </div>
27     <% end %>
28   </div>
```

そして、lib/nano_planner_web/templates/plan_item ディレクトリに新規ファイル _index_row.html.eex を作成して、さきほど切り取った内容を貼り付けます。

lib/nano_planner_web/templates/plan_item/_index_row.html.eex (New)

```
 1   <div class="row">
 2     <div class="col-8 col-md-4">
 3       <span class="plan-item-name">
 :
20     <div class="col-12">
21       <span class="plan-item-duration">
22         <%= format_duration(item) %>
23       </span>
24     </div>
25   </div>
```

貼り付けた後で、ソースコード全体のインデントを半角スペース 2 個分減らしています。

ファイル _index_row.html.eex に含まれる変数 item をすべて @item で置き換えます（6 ヶ所）。

lib/nano_planner_web/templates/plan_item/_index_row.html.eex

```
1   <div class="row">
2     <div class="col-8 col-md-4">
```

```
3       <span class="plan-item-name">
4 -       <%= link item.name, to: Routes.plan_item_path(@conn, :show, item.id) >
    %>
4 +       <%= link @item.name, to: Routes.plan_item_path(@conn, :show, @item.>
    id) %>
5       </span>
6     </div>
7     <div class="col-4 d-md-none text-right toolbar">
8 -     <%= render "_xs_toolbar.html", conn: @conn, item: item %>
8 +     <%= render "_xs_toolbar.html", conn: @conn, item: @item %>
9     </div>
10    <div class="col-12 col-md-5">
11      <span class="plan-item-description">
12 -       <%= for line <- String.split(item.description, "\n") do %>
12 +       <%= for line <- String.split(@item.description, "\n") do %>
13          <%= line %><br>
14        <% end %>
15      </span>
16    </div>
17    <div class="col-3 d-none d-md-block text-right toolbar">
18 -     <%= render "_md_toolbar.html", conn: @conn, item: item %>
18 +     <%= render "_md_toolbar.html", conn: @conn, item: @item %>
19    </div>
20    <div class="col-12">
21      <span class="plan-item-duration">
22 -       <%= format_duration(item) %>
22 +       <%= format_duration(@item) %>
23      </span>
24    </div>
25  </div>
```

このファイルを部分テンプレートとして埋め込むように切り取り元の EEx テンプレートを書き換えます。

lib/nano_planner_web/templates/plan_item/index.html.eex

```
:
5   <div class="container-fluid plan-items">
6     <%= for item <- @plan_items do %>
7 +     <%= render "_index_row.html", item: item, conn: @conn %>
```

```
8     <% end %>
9   </div>
```

EEx テンプレートの作成

of_today アクションで使用する EEx テンプレートを次のような内容で作成します。

lib/nano_planner_web/templates/plan_item/of_today.html.eex (New)

```
1   <%= render "index.html", plan_items: @plan_items, conn: @conn %>
2
3   <hr>
4
5   <div class="container-fluid plan-items">
6     <%= for item <- @continued_plan_items do %>
7       <%= render "_index_row.html", item: item, conn: @conn %>
8     <% end %>
9   </div>
```

1 行目では index アクション用のテンプレートを部分テンプレートとして利用しています。5 行目以降が「継続中の今日の予定表」を表示するコードです。あらかじめ for ブロックの内側を部分テンプレートとして切り出しておいたため、短いコードを追加するだけで済みました。

3 行目では HTML の hr 要素を用いて「狭義の今日の予定表」と「継続中の今日の予定表」の間に水平線を表示しています。

シードデータの書き換えと動作確認

では、動作確認をするため「継続中の予定項目」が登録されるようにシードデータを書き換えましょう。

priv/repo/seeds.exx

```
  :
35  insert!(%PlanItem{
```

```
36      name: "姉の出張",
37      description: "札幌",
38      starts_at: Timex.shift(time0, days: -1, hours: 10),
39 -    ends_at: Timex.shift(time0, hours: 17)
39 +    ends_at: Timex.shift(time0, days: 1, hours: 17)
40    })
```

Phoenix サーバーを止めてから、`mix ecto.reset` コマンドでシードデータを投入し直し、Phoenix サーバーを起動して、ブラウザで「今日の予定表」を表示すると図 7.2 のような画面になります。

図 7.2　水平線の下に継続中の予定項目を表示

関数 Enum.empty?/1

さて、現行の実装では「継続中の予定項目」の件数が 0 であるときにも、「狭義の今日の予定表」と「継続中の今日の予定表」の間に水平線が表示されてしまいます。これを防ぐには、追加したコード全体を unless ... end で囲みます。

lib/nano_planner_web/templates/plan_item/of_today.html.eex

```
1   <%= render "index.html", plan_items: @plan_items, conn: @conn %>
2
3 - <hr>
4 -
5 - <div class="container-fluid plan-items">
6 -   <%= for item <- @continued_plan_items do %>
7 -     <%= render "_index_row.html", item: item, conn: @conn %>
8 -   <% end %>
9 - </div>
3 + <%= unless Enum.empty?(@continued_plan_items) do %>
4 +   <hr>
5 +
6 +   <div class="container-fluid plan-items">
7 +     <%= for item <- @continued_plan_items do %>
8 +       <%= render "_index_row.html", item: item, conn: @conn %>
9 +     <% end %>
10 +  </div>
11 + <% end %>
```

関数 Enum.empty?/1 は引数にリストやマップなどを取り、中身が空であるときに true を返します。

■コラム: 連結リスト

テンプレート of_today.html.eex の 3 行目をご覧ください。

```
<%= unless Enum.empty?(@continued_plan_items) do %>
```

ここは次のようにも書けます。

```
<%= if length(@continued_plan_items) > 0 do %>
```

Kernel モジュールの関数 length/1 は、リストを引数に取ってその要素数を返します。したがって、このように書いても水平線の表示・非表示を制御できます。しかし、まったく同じではありません。

Elixir のリストは**連結リスト**（linked list）として実装されているため、リストの要素数を取得するには先頭から末尾まですべての要素を順にたどっていかなければなりません。そのため、サイズの大きなリストの要素数を取得するには時間がかかります。ここでは正確な要素数を知りたいわけではなく、空かどうかを調べたいだけです。関数 Enum.empty?/1 はリストの先頭の要素を取って、それが nil であれば true を返し、さもなくば false を返します。つまり、このケースでは関数 Kernel.length/1 よりも関数 Enum.empty?/1 の方が効率が良いのです。

NanoPlanner の常識的な利用法において @continued_plan_items が巨大なリストになることは考えにくいので、あまり神経質になる必要はありません。しかし、Elixir のリストの重要な特徴として記憶にとどめてください。