第 16 章

文字列の長さに関するバリデーション

この章では、文字列の長さがある条件を満たすことを検証する方法について学びます。そして、「件名」フィールドと「説明」フィールドに関して文字列の長さがある限度を超えないように制限する機能を NanoPlanner に加えます。

16.1 さまざまなバリデーション関数

主なバリデーション関数

第 13 章で、予定項目の件名フィールド（name 属性）が空でないことを確認するため、PlanItem.changeset/2 関数のソースコードに次のような記述を追加しました。

```
    |> validate_required([:name])
```

パイプ演算子を通じて流れくるのはチェンジセットです。そのチェンジセットの name 属性を示すアトム :name を含むリストを Ecto.Changeset モジュールの関数 validate_required/3 に渡しています。

Ecto.Changeset モジュールには "validate_" で始まる名前を持つ 10 個あまりの関数が定義されています。主なものを表 16.1 にまとめました。

表 16.1　バリデーション関数のリスト

関数名/アリティ	検証する内容	エラーメッセージ ID
validate_acceptance/3	true である	must be accepted
validate_exclusion/4	リストに含まれない	is reserved
validate_format/4	形式に合致する	has invalid format
validate_inclusion/4	リストに含まれる	is invalid
validate_length/3	長さ・要素数が条件を満たす	表 16.2 を参照
validate_number/3	数値が条件を満たす	表 16.3 を参照
validate_required/3	空でも nil でもない	can't be blank

本書ではこれらのバリデーション関数をひとつずつ説明することはしません。すでに説明した関数 validate_required/3 の他、関数 validate_length/3 と関数 validate_number/3 についてのみ概要を説明します。バリデーション関数全般については、次の URL にある公式ドキュメント（英語）を参照してください。

https://hexdocs.pm/ecto/Ecto.Changeset.html#functions

関数 validate_length/3

関数 validate_length/3 は、文字列の長さおよびリストの要素数がある条件を満たすことを検証します。条件を示すためのオプションとして is（等しい）、max（以下）、min（以上）が使えます。例えば、次のように記述すれば、code パラメータの文字列としての長さが 10 を超えないことを検証します。

```
    |> validate_length(:code, max: 10)
```

もし、長さが 4 から 10 の間にあることを確かめたければ、次のようにオプション min を加えます。

```
    |> validate_length(:code, min: 4, max: 10)
```

関数 validate_length/3 のエラーメッセージ ID は違反した条件により異なります（表 16.2）。

表 16.2　関数 validate_length/3 のエラーメッセージ ID

値の型	条件	エラーメッセージ ID
文字列	長さが count に等しい	should be %{count} character(s)
文字列	長さが count 以上	should be at least %{count} character(s)
文字列	長さが count 以下	should be at most %{count} character(s)
リスト	個数が count に等しい	should be %{count} item(s)
リスト	個数が count 以上	should be at least %{count} item(s)
リスト	個数が count 以下	should be at most %{count} item(s)

関数 validate_number/3

関数 validate_number/3 は、数値（整数および浮動小数点数）がある条件を満たすことを検証します。条件を示すためのオプションとして less_than（未満）、greater_than（を超える）、less_than_or_equal_to（以下）、greater_than_or_equal_to（以上）、equal_to（に等しい）が使えます。例えば、次のように記述すれば、price パラメータの数値が 0 以上であることを検証します。

```
|> validate_number(:price, greater_than_or_equal_to: 0)
```

関数 validate_number/3 のエラーメッセージ ID は違反した条件により異なります（表 16.3）。

表 16.3　関数 validate_number/3 のエラーメッセージ ID

条件	エラーメッセージ ID
number 未満	should be less than %{number}
number を超える	should be greater than %{number}
number 以下	should be less than or equal to %{number}
number 以上	should be greater than or equal to %{number}
number に等しい	should be equal to %{number}

数値同士の比較には decimal パッケージの関数 Decimal.cmp/2 が使用されます。decimal パッケージは ecto モジュールによりインストールされます。関数 Decimal.cmp/2 には、型の異なる数値の比較ができるという特長があります。引数に "3.14" のような文字列と整数や浮動小数点数を比較することもできます。

16.2 件名の文字数を制限する

PlanItem モジュールの書き換え

では、実際に関数 validate_length/3 を使ってみましょう。PlanItem モジュール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/schedule/plan_item.ex

```
  :
45    def changeset(%PlanItem{} = plan_item, %{"all_day" => "false"} = attrs) do
46      plan_item
47      |> cast(attrs, @common_fields ++ @date_time_fields)
48      |> change_starts_at()
49      |> change_ends_at()
50      |> validate_required([:name])
51 +    |> validate_length(:name, max: 80)
52    end
53
54    def changeset(%PlanItem{} = plan_item, %{"all_day" => "true"} = attrs) do
55      plan_item
56      |> cast(attrs, @common_fields ++ @date_fields)
57      |> change_time_boundaries()
58      |> validate_required([:name])
59 +    |> validate_length(:name, max: 80)
60    end
61
62    def changeset(%PlanItem{} = plan_item, attrs) do
63      plan_item
64      |> cast(attrs, @common_fields)
65      |> validate_required([:name])
66 +    |> validate_length(:name, max: 80)
```

```
67      end
 :
```

予定項目の件名の最大文字数を 80 に設定しています。コードの重複が気になるところですが、いったん良しとしましょう。あとでリファクタリングを行います。

続いて、日本語用の PO ファイル errors.po にエラーメッセージの翻訳を付け加えます。

priv/gettext/ja/LC_MESSAGES/errors.po

```
 :
60    msgid "should be at most %{count} character(s)"
61    msgid_plural "should be at most %{count} character(s)"
62 -  msgstr[0] ""
62 +  msgstr[0] "は%{count}文字以内で入力してください"
 :
```

文字列の長さが上限を越えたときに関数 validate_length/3 が使用するメッセージ ID は

```
should be at most %{count} character(s)
```

です。メッセージ ID に %{count}キーが含まれる場合、その値に基づいて翻訳の形を変化させるために複数個の翻訳を与える必要があります。用意すべき翻訳の個数はロケールにより異なります。日本語は 1、英語やフランス語は 2、ロシア語は 3、アラビア語は 6 です。

PO ファイルの msgstr[0] で始まる行には第 1 の翻訳を記述します。そして、msgstr[1] で始まる行には第 2 の翻訳……と続きます。日本語用の PO ファイルでは各メッセージ ID に対して msgstr[0] の行しか存在しません。しかし、priv/gettext/en/LC_MESSAGES ディレクトリにある英語用の errors.po を見ると、msgstr[0] と msgstr[1] の行が存在します。

```
msgid "should be at most %{count} character(s)"
msgid_plural "should be at most %{count} character(s)"
msgstr[0] ""
msgstr[1] ""
```

PO ファイルの msgid_plural で始まる行に書かれているのは第 2 メッセージ ID です。第 15 章「国際化と地域化」でも触れましたが、第 1 と第 2 のメッセージ ID は同じであることもあります。

では、動作確認をしましょう。ブラウザで予定項目の追加フォームを開き、件名に 80 文字を超える長さの文字列を入力して送信し、図 16.1 のようにエラーがメッセージが表示されれば OK です。

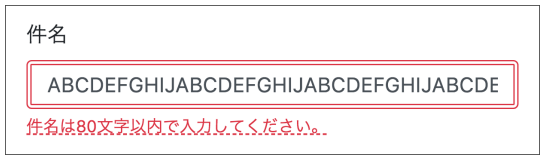

図 16.1　件名の長さに関するエラーメッセージが表示された

共通コードの抽出

さて、PlanItem モジュールでは changeset という名前の関数が 3 パターン定義されていて、いずれにも次のようなコードが含まれています。

```
    |> validate_required([:name])
    |> validate_length(:name, max: 80)
```

この重複を除去しましょう。まず、次のようにプライベート関数 validate_common_fields/1 を追加します。

lib/nano_planner/schedule/plan_item.ex

```
  :
108     defp time_zone do
109       Application.get_env(:nano_planner, :default_time_zone)
110     end
111 +
112 +   defp validate_common_fields(changeset) do
113 +     changeset
114 +     |> validate_required([:name])
115 +     |> validate_length(:name, max: 80)
```

```
116 +   end
117   end
```

そして、このプライベート関数を用いて、PlanItem モジュールのリファクタリングを行います。

lib/nano_planner/schedule/plan_item.ex

```
  :
45    def changeset(%PlanItem{} = plan_item, %{"all_day" => "false"} = attrs) do
46      plan_item
47      |> cast(attrs, @common_fields ++ @date_time_fields)
48      |> change_starts_at()
49      |> change_ends_at()
50 -    |> validate_required([:name])
51 -    |> validate_length(:name, max: 80)
50 +    |> validate_common_fields()
51    end
52
53    def changeset(%PlanItem{} = plan_item, %{"all_day" => "true"} = attrs) do
54      plan_item
55      |> cast(attrs, @common_fields ++ @date_fields)
56      |> change_time_boundaries()
57 -    |> validate_required([:name])
58 -    |> validate_length(:name, max: 80)
57 +    |> validate_common_fields()
58    end
59
60    def changeset(%PlanItem{} = plan_item, attrs) do
61      plan_item
62      |> cast(attrs, @common_fields)
63 -    |> validate_required([:name])
64 -    |> validate_length(:name, max: 80)
63 +    |> validate_common_fields()
64    end
  :
```

16.3 「説明」フィールドのバリデーション

プライベート関数 validate_common_fields/1 の書き換え

第 1 章で定めた予定項目に関するバリデーションの仕様によれば、説明の最大文字数は 400 です。フォームから送信された description パラメータの値がこの条件に合致することを検証しましょう。前節で抽出したプライベート関数 PlanItem.validate_common_fields/1 のコードを次のように書き換えてください。

lib/nano_planner/schedule/plan_item.ex

```
  :
109     defp validate_common_fields(changeset) do
110       changeset
111       |> validate_required([:name])
112       |> validate_length(:name, max: 80)
113 +     |> validate_length(:description, max: 400)
114     end
115   end
```

ヘルパー関数 bootstrap_textarea/3 の作成

次に、説明を入力するためのテキストエリアの下にエラーメッセージが表示されるようにします。エラーメッセージは Bootstrap を使ってスタイリングします。BootstrapHelpers モジュールに新たな関数 bootstrap_textarea/3 を次のように追加してください。

lib/nano_planner_web/views/bootstrap_helpers.ex

```
  :
 7     def bootstrap_text_input(form, field, opts \\ []) do
 8       class = form_control_class(form, field, opts)
 9       opts = Keyword.put(opts, :class, class)
10
11       text_input(form, field, opts)
```

```
12      end
13 +
14 +    def bootstrap_textarea(form, field, opts \\ []) do
15 +      class = form_control_class(form, field, opts)
16 +      opts = Keyword.put(opts, :class, class)
17 +
18 +      textarea(form, field, opts)
19 +    end
20
21      defp form_control_class(form, field, opts) do
 :
```

この関数は直前の関数 bootstrap_text_input/3 と非常によく似ています。すぐ後でリファクタリングを行います。

続いて、予定項目の追加・変更フォームの HTML テンプレートでこの関数を利用します。

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

```
 :
 6    <div class="form-group">
 7      <%= label @f, :description, dgettext("schema", "plan_item|description") %>
 8 -    <%= textarea @f, :description, class: "form-control", rows: 4 %>
 8 +    <%= bootstrap_textarea @f, :description, rows: 4 %>
 9 +    <%= bootstrap_feedback @f, :description %>
10    </div>
 :
```

ブラウザで予定項目の追加フォームを開き、説明入力欄に 400 文字を超える長さのテキストを入力して送信すると図 16.2 のようにエラーメッセージが表示されます。

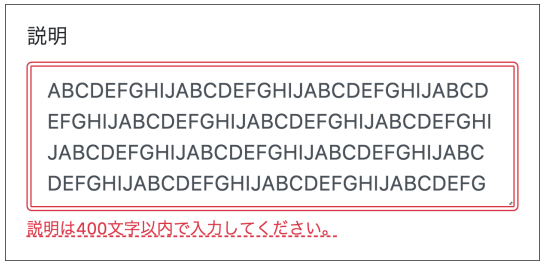

図 16.2　説明の長さに関するエラーメッセージが表示された

共通コードの抽出

最後に、関数 bootstrap_text_input/3 と関数 bootstrap_textarea/3 にあるコードの重複を解消しましょう。

まず、重複部分を抜き出してプラベート関数 html_opts/3 を作ります。

lib/nano_planner_web/views/bootstrap_helpers.ex

```
  :
 20 +
 21 +   defp html_opts(form, field, opts) do
 22 +     class = form_control_class(form, field, opts)
 23 +     Keyword.put(opts, :class, class)
 24 +   end
 25
 26    defp form_control_class(form, field, opts) do
  :
```

そして、この関数を利用して関数 bootstrap_text_input/3 を次のように書き換えます。

lib/nano_planner_web/views/bootstrap_helpers.ex

```
  :
  7    def bootstrap_text_input(form, field, opts \\ []) do
  8 -    class = form_control_class(form, field, opts)
  9 -    opts = html_opts(form, field, opts)
 10 -
```

```
11 -      text_input(form, field, opts)
 8 +      text_input(form, field, html_opts(form, field, opts))
 9      end
10
 :
```

同様に、関数 bootstrap_textarea/3 を次のように書き換えます。

lib/nano_planner_web/views/bootstrap_helpers.ex

```
 :
11      def bootstrap_textarea(form, field, opts \\ []) do
12 -      class = form_control_class(form, field, opts)
13 -      opts = Keyword.put(opts, :class, class)
14 -
15 -      textarea(form, field, opts)
12 +      textarea(form, field, html_opts(form, field, opts))
13      end
 :
```

ブラウザで動作確認をしてから次の章に進んでください。