

Laravel 資料 4 バリデーション

導入

本資料では Laravel でのバリデーションとフラッシュメッセージについて、資料 2 と 3 で作成したログインページに機能を実装しながら説明を行う。

以下の手順で行っていく。

1. 用語説明
2. バリデーション実装
 1. validate メソッドによるバリデーション
 2. ビューの変更（共通）
 3. バリデータ生成によるバリデーション
 4. フォームリクエスト生成によるバリデーション
 5. エラーメッセージの日本語化
3. 入力値の保持

バリデーションとは

バリデーション（Validation [発音記号：v`ælədéiʃən] ）とは批准、確認という意味で、プログラミングにおいては入力値のチェックを意味する。

ユーザが入力した内容が要件（入力形式）を満たしているか、必須入力の項目に抜けがないか、不正な値が入力されていないかといったことをチェックする機能である。

■ Laravel におけるバリデーション

入力されたリクエストパラメータに対し、バリデーションで設定したルールに基づいてチェックを行う。

すべてのチェックが成功するとバリデーション以降の処理が実行され、失敗した場合は例外が投げられ、自動的に前のページへリダイレクトする。

また、バリデーションエラーはフラッシュデータとして保存される。

フラッシュデータとは、直後の HTTP リクエスト（ページ遷移と思えばよい）の間だけセッションに保存されるデータのことである。

エラーメッセージは `$errors` という変数名でビューに渡される。

バリデーションの実装

本資料では 3 通りのバリデーション方法を説明する。

1. validate メソッド
2. バリデータの生成
3. フォームリクエストの生成

ビュー側は共通であるため validate メソッドの項目でのみ説明する。

どの方法についても資料 3 の最終段階に対して、以下の変更を加えた後の差分を示す。

【app\Http\Controllers\LoginController.php】

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;

class LoginController extends Controller
{
    public function getIndex()
    {
        (略)
    }

    public function postIndex(Request $request)
    {
        // リクエストパラメータを配列として全件取得
        $input = $request->all();

        -
        - // 権限はNOT NULLのため入力されてなければすぐ弾いていい
        - if (!isset($input['authority'])) {
        -     return view('login/login');
        - }

        // DBのデータと照合
        $db_result = User::where('name', $input['name'])
            ->where('password', $input['password'])
            ->where('authority', $input['authority'])
            ->get();

        // 一致するデータなし
        if (count($db_result) == 0) {
            return view('login/login');
        }

        return view('login/result')->with('input', $input);
    }
}
```

バリデーションで権限（authority）は必須入力とするため、権限の NULL 判定は不要になるため削除する。

1.validate メソッド

■ 1-1.コントローラの変更

`Illuminate\Http\Request` オブジェクトの `validate` メソッドを使用する。

【app\Http\Controllers\LoginController.php】

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;

class LoginController extends Controller
{
    public function getIndex()
    {
        (略)
    }

    public function postIndex(Request $request)
    {
        + // Validation
        + $rules = [
        +     'name' => 'required',
        +     'password' => 'required',
        +     'authority' => 'required|integer|min:1|max:2',
        + ];
        + $this->validate($request, $rules);
        +

        // リクエストパラメータを配列として全件取得
        $input = $request->all();

        // DBのデータと照合
        $db_result = User::where('name', $input['name'])
            ->where('password', $input['password'])
            ->where('authority', $input['authority'])
            ->get();

        // 一致するデータなし
        if (count($db_result) == 0) {
            return view('login/login');
        }

        return view('login/result')->with('input', $input);
    }
}
```

`validate` メソッドの第 1 引数にリクエストオブジェクト、第 2 引数に連想配列でバリデーションルールを渡している。

例えば '`required`' は必須入力であることを示す。

その他バリデーションルールについては公式ドキュメントを参照するとよい。

(公式) 使用可能なバリデーションルール [🔗](#)

■ 1-2. ビューの変更 (まとめて表示)

バリデーションに失敗した際にエラーメッセージを表示するよう、ログイントップページのビューを変更する。

下記は上部にまとめてエラーメッセージを表示する方法である。

`any` メソッドでエラーメッセージの存在確認を行い、その後 `all` メソッドで全フィールドの全エラーメッセージを取得して表示している。

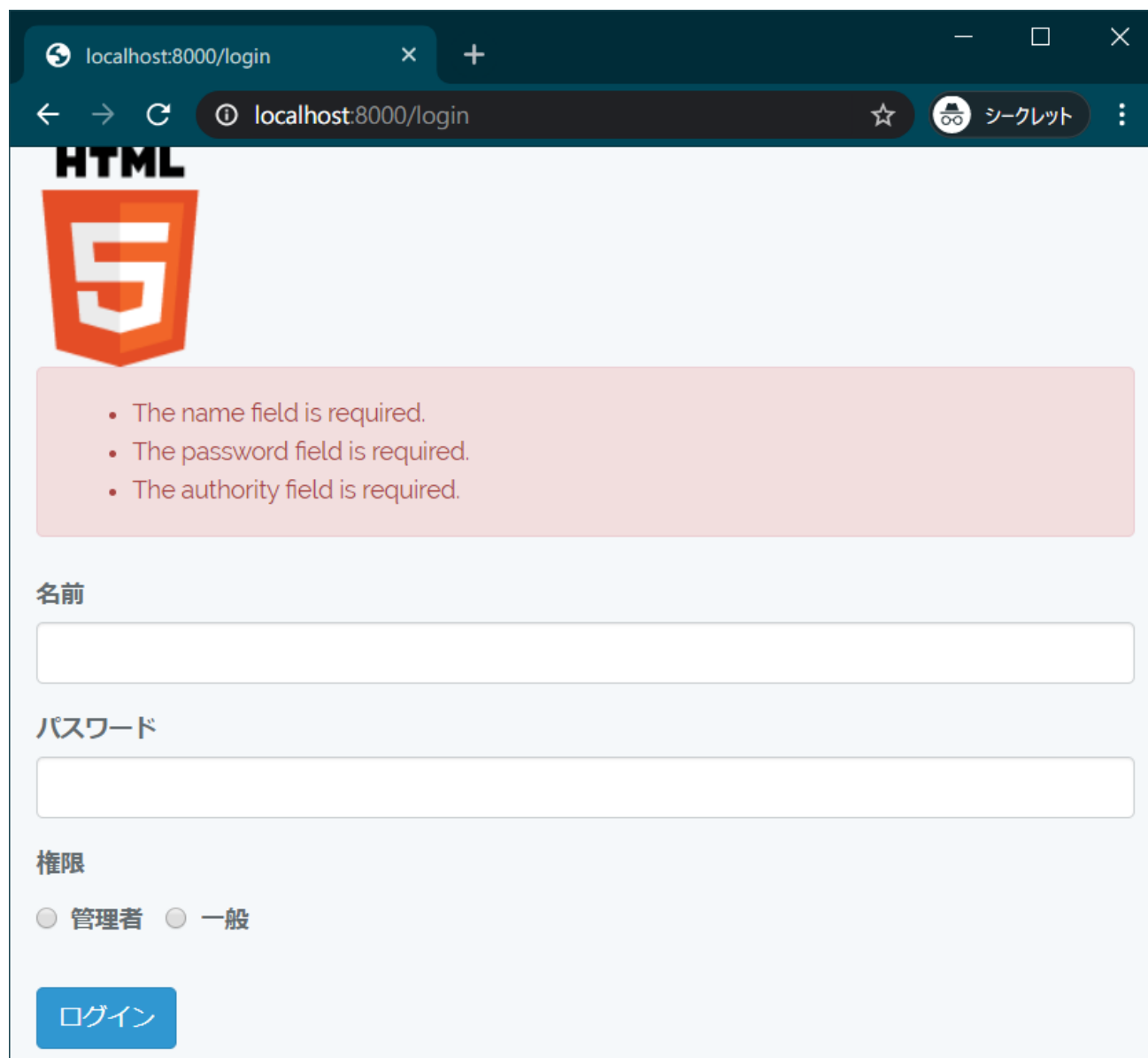
【resources\views\login\login.blade.php】

```
@extends('layout/layout')
@section('content')
<form method="post" action="/login">
    <h2>ログイン</h2>
+   @if ($errors->any())
+   <div class="alert alert-danger">
+       <ul>
+       @foreach ($errors->all() as $error)
+           <li>{{ $error }}</li>
+       @endforeach
+       </ul>
+   </div>
+   @endif
    {{ csrf_field() }}
```

(略)

■ 1-3.動作確認

何も入力せず「ログイン」ボタンを押下すると、再びログイントップページに遷移し、以下のようなエラーメッセージが表示される。



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/login'. The page features the HTML5 logo at the top left. Below the logo, a pink error message box contains the following text:

- The name field is required.
- The password field is required.
- The authority field is required.

Below the error messages, the form includes the following fields and controls:

- 名前** (Name): A text input field.
- パスワード** (Password): A password input field.
- 権限** (Authority): Two radio button options, '管理者' (Administrator) and '一般' (General).
- ログイン** (Login): A blue button.

英語で必須入力のバリデーションエラーメッセージが表示されている。

■ 1-4.ビューの変更（個別表示）

エラーメッセージを要素ごと個別に表示するようにビューを変更する。

`first`メソッドは指定したフィールドの最初のエラーメッセージを取得する（バリデーションルールが複数ある場合、複数のエラーメッセージが返されることはあり得る）。その結果を見てエラー用のクラスを付けたり、個別にメッセージを表示したりしている。

【resources\views\login\login.blade.php】

```
(略)
<label>名前</label>
<div class="form-group">
-   <input type="text" name="name" class="form-control">
+   <input type="text" name="name"
+       class="form-control @if(!empty($errors->first('name'))border-danger
@elseif">
+   <p>
+       <span class="help-block text-danger">{{$errors->first('name')}}</span>
+   </p>
</div>
<label>パスワード</label>
<div class="form-group ">
-   <input type="password" name="password" class="form-control">
+   <input type="password" name="password"
+       class="form-control @if(!empty($errors->first('name'))border-danger
@elseif">
+   <p>
+       <span class="help-block text-danger">
+           {{$errors->first('password')}}
+       </span>
+   </p>
</div>
<label>権限</label>
- <div class="form-group">
+ <div class="form-group
+ @if(!empty($errors->first('authority'))text-danger @elseif">
  <div class="radio-inline">
    <label>
      <input type="radio" name="authority" value="1">管理者
    </label>
  </div>
  <div class="radio-inline">
    <label>
      <input type="radio" name="authority" value="2">一般
    </label>
  </div>
+   <p>
+       <span class="help-block text-danger">{{$errors->first('authority')}}
</span>
+   </p>
</div>
```

(略)

■ 1-5.動作確認

何も入力せず「ログイン」ボタンを押下すると、以下のようなエラーメッセージが表示される。



The screenshot shows a web browser window with the address bar displaying "localhost:8000/login". The page features the HTML5 logo at the top. Below the logo, a red error message box contains the following text:

- The name field is required.
- The password field is required.
- The authority field is required.

The form consists of three sections:

- 名前** (Name): A text input field with a red border and the message "The name field is required." below it.
- パスワード** (Password): A text input field with a red border and the message "The password field is required." below it.
- 権限** (Authority): Two radio buttons labeled "管理者" (Administrator) and "一般" (General), with the message "The authority field is required." below them.

A blue "ログイン" (Login) button is located at the bottom left of the form.

2.バリデータの生成

■ 2-1.コントローラの変更

Validator ファサードを使い、バリデータインスタンスを**make**メソッドで生成する。

【app\Http\Controllers\LoginController.php】

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
+ use Validator;

class LoginController extends Controller
{
    public function getIndex()
    {
        (略)
    }

    public function postIndex(Request $request)
    {
        // リクエストパラメータを配列として全件取得
        $input = $request->all();

+
+         // Validation
+         $rules = [
+             'name' => 'required',
+             'password' => 'required',
+             'authority' => 'required|integer|min:1|max:2',
+         ];
+         $this->validate($input, $rules);

        // DBのデータと照合
        $db_result = User::where('name', $input['name'])
            ->where('password', $input['password'])
            ->where('authority', $input['authority'])
            ->get();

        // 一致するデータなし
        if (count($db_result) == 0) {
            return view('login/login');
        }

        return view('login/result')->with('input', $input);
    }
}
```

make メソッドの第1引数にリクエストパラメータ（連想配列）、第2引数に連想配列でバリデーションルールを渡している。

■ 2-2.動作確認

1-5 の動作確認と同様の結果になることを確認する。

何も入力せず「ログイン」ボタンを押下すると、以下のようなエラーメッセージが表示される。



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/login'. The page features the HTML5 logo at the top. Below the logo, a red box contains three error messages: 'The name field is required.', 'The password field is required.', and 'The authority field is required.'.

Below the error messages, the form fields are labeled in Japanese:

- 名前** (Name): A text input field with a red border and the error message 'The name field is required.' below it.
- パスワード** (Password): A text input field with a red border and the error message 'The password field is required.' below it.
- 権限** (Authority): Two radio buttons labeled **管理者** (Administrator) and **一般** (General). The **一般** option is selected. Below the radio buttons is the error message 'The authority field is required.'

At the bottom of the form is a blue button labeled **ログイン** (Login).

■ 2-3.エラーメッセージの日本語化（一部）

`make`メソッドは第 3 引数にカスタムエラーメッセージを渡すことができる。

`postIndex`メソッドに以下のように追記する。

【app\Http\Controllers\LoginController.php】

```
(略)
public function postIndex(Request $request)
{
    // リクエストパラメータを配列として全件取得
    $input = $request->all();

    // Validation
    $rules = [
        'name' => 'required',
        'password' => 'required',
        'authority' => 'required|integer|min:1|max:2',
    ];
+   $error_msg = [
+       'required' => ':attributeは必須入力です。'
+   ];
-   Validator::make($input, $rules)->validate();
+   Validator::make($input, $rules, $error_msg)->validate();

    // DBのデータと照合
    $db_result = User::where('name', $input['name'])
        ->where('password', $input['password'])
        ->where('authority', $input['authority'])
        ->get();

    // 一致するデータなし
    if (count($db_result) == 0) {
        return view('login/login');
    }

    return view('login/result')->with('input', $input);
}
(略)
```

権限の必須入力以外のバリデーションルールは確認しづらいため、今回は `required` のエラーメッセージのみ記述した。

`:attribute`はプレースホルダであり、バリデーション対象のフィールドの名前（input タグの `name`）に置換される。



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/login'. The page has a dark blue header with the text 'HTML' and a large orange and white logo. Below the logo, there is a pink box containing three bullet points in Japanese: 'nameは必須入力です。', 'passwordは必須入力です。', and 'authorityは必須入力です。'. The main content area has a light blue background. It contains three sections: '名前' (Name) with an empty input field and the error message 'nameは必須入力です。'; 'パスワード' (Password) with an empty input field and the error message 'passwordは必須入力です。'; and '権限' (Authority) with two radio buttons labeled '管理者' (Administrator) and '一般' (General), and the error message 'authorityは必須入力です。'. At the bottom left, there is a blue button labeled 'ログイン' (Login).

何も入力せず「ログイン」ボタンを押下すると、以下のようなエラーメッセージが表示される。inputタグの name 要素で置換しているため、要素名は日本語化できていない。
:attributeも日本語化する方法は次の項目で説明する。

3. フォームリクエストの生成

この方法の場合、複数ページに対するバリデーションの設定を一括で行える。

そのため、基本的にはフォームリクエストの生成でバリデーションは行い、コントローラ内に記述する2つの方法は一部違うページなど、例外への対応として使用するのがよいだろう。

■ 3-1. フォームリクエスト作成

shell を起動し（コマンドプロンプトや VSCode のターミナルでもよい）、`LaravelSample`ディレクトリ下で以下のコマンドを実行する。

```
php artisan make:request LoginRequest
```

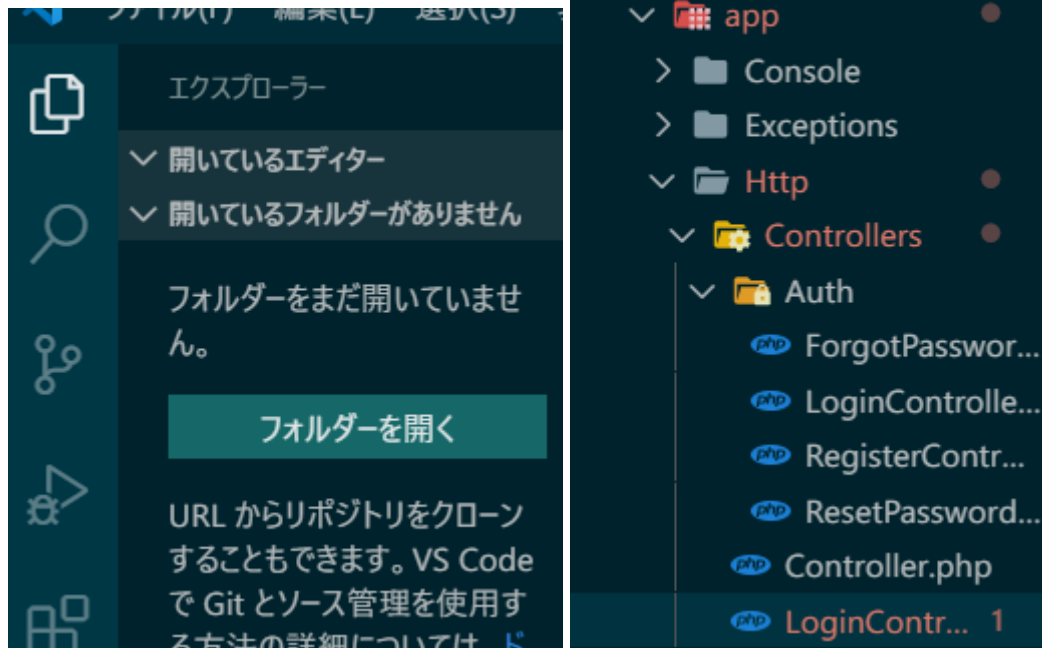
「LoginRequest」が作成するフォームリクエストの名前である。

命名規則としてはアッパーキャメルケースなだけで、必ずしも「Request」と名前に付ける必要はない。

生成されたリクエストクラスは`app/Http/Request`ディレクトリに設置される。

ところで VSCode にはワークスペースという概念がある。

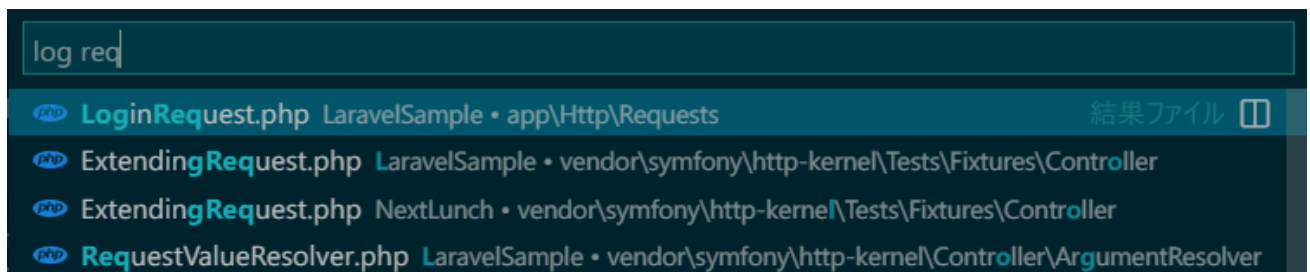
細かいことは分からなくてよいが、以下の「フォルダーを開く」から作業フォルダを開くか、この場所にフォルダをドラッグ&ドロップすると、そのフォルダがワークスペースとして設定され、フォルダツ



リーが表示される。

これだけではさほど便利さは感じないが、ワークスペースに設定していると、**Ctrl+p**のショートカットキーで開く入力欄からファイル名を検索して開くことができる。

手作業で探すよりもよほど早いため、活用することをおすすめする。



■ 3-2. リクエストクラスの変更

作成したリクエストクラスを有効化し、ルールを追加する。

【app\Http\Requests\LoginRequest.php】

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class LoginRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
-         return false;
+         return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
-             //
+             'name' => 'required',
+             'password' => 'required',
+             'authority' => 'required|integer|min:1|max:2',
        ];
    }
}
```

■ 3-3.コントローラの変更

`postIndex`メソッドで受取るリクエストのタイプヒントを自作したリクエストクラスに変更する。これによってフォームから送られてきたリクエストは`LoginRequest.php`で設定したルールに基づいてバリデーションされた後、`postIndex`メソッドが実行されることになる。

【app\Http\Controllers\LoginController.php】

```
<?php

namespace App\Http\Controllers;

- use Illuminate\Http\Request;
  use App\Models\User;
+ use App\Http\Requests\LoginRequest;

class LoginController extends Controller
{
    public function getIndex()
    {
        (略)
    }

-   public function postIndex(Request $request)
+   public function postIndex(LoginRequest $request)
    {
        // リクエストパラメータを配列として全件取得
        $input = $request->all();

        // DBのデータと照合
        $db_result = User::where('name', $input['name'])
            ->where('password', $input['password'])
            ->where('authority', $input['authority'])
            ->get();

        // 一致するデータなし
        if (count($db_result) == 0) {
            return view('login/login');
        }

        return view('login/result')->with('input', $input);
    }
}
```

実質 `use` の行が増えただけである。

コントローラに処理を書きすぎると読みづらくなるめ、分けられるものはなるべく分けて書くとよい。

また、前述の通り登録と更新のような同様の入力項目のページに対し、リクエストクラスのタイプヒントを変更するのみで同じバリデーションを設定できる点もこの方法の利点である。

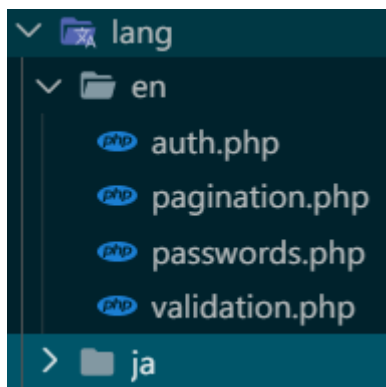
■ 3-4. エラーメッセージの日本語化

3-3 の動作確認の説明は 1-5、2-2 と同じ結果のため省略する。

この項目では言語ファイルの設定を変更し、日本語化を行う。

3-4-1. 言語設定ファイルの作成

`resources\lang` ディレクトリに移動すると、英語の言語設定フォルダ「en」が存在する。
同じ階層に日本語用の「ja」フォルダを作成する。



今回は github から日本語用設定ファイルをダウンロードする。
ターミナルで ja ディレクトリに移動し、以下のコマンドを実行する。

```
curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/auth.php
curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/pagination.php
curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/passwords.php
curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/validation.php
```

```
nextthought@LAPTOP-GQOJH1IC MINGW64 /c/xampp/htdocs/LaravelSample/resources/lang/ja
$ curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/auth.php
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100   915  100   915    0     0  1756      0 --:--:-- --:--:-- --:--:-- 1756

nextthought@LAPTOP-GQOJH1IC MINGW64 /c/xampp/htdocs/LaravelSample/resources/lang/ja
$ curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/pagination.php
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100   571  100   571    0     0  1297      0 --:--:-- --:--:-- --:--:-- 1297

nextthought@LAPTOP-GQOJH1IC MINGW64 /c/xampp/htdocs/LaravelSample/resources/lang/ja
$ curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/passwords.php
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100   912  100   912    0     0  2876      0 --:--:-- --:--:-- --:--:-- 2876

nextthought@LAPTOP-GQOJH1IC MINGW64 /c/xampp/htdocs/LaravelSample/resources/lang/ja
$ curl -OL https://raw.githubusercontent.com/rito-nishino/Laravel-Japanese-Language-fileset/master/validation.php
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left   Speed
100  9827  100  9827    0     0 30613      0 --:--:-- --:--:-- --:--:-- 30613
```

すべてダウンロードが終了したら、念の為ファイルを ja フォルダに確認しに行く。ダウンロードした `auth.php`、`pagination.php`、`passwords.php`、`validation.php` が存在していれば問題ない。

なお、各ファイルは以下のような記述になっている。

2-3 で日本語化した際と同様に連想配列で書かれていることがわかる。

```
<?php
return [
    ...../*
    .....*/
    .....- Validation Language Lines
    .....- 検証言語
    .....-
    .....-
    .....- The following language lines contain the default error messages used by
    .....- the validator class. Some of these rules have multiple versions such
    .....- as the size rules. Feel free to tweak each of these messages here.
    .....-
    .....- 次の言語行には、バリデータークラスで使用するデフォルトのエラーメッセー
    .....- ジが含まれています。
    .....- これらの規則の中には、サイズ規則などの複数のバージョンがあります。
    .....- これらのメッセージのそれぞれをここで微調整してください。
    .....*/
    .....'accepted'=>':attribute: が未承認です',
    .....'active_url'=>':attribute: は有効なURLではありません',
    .....'after'=>':attribute: は :date: より後の日付にしてください',
    .....'after_or_equal'=>':attribute: は :date: 以降の日付にしてください',
    .....'alpha'=>':attribute: は英字のみ有効です',
    .....'alpha_dash'=>':attribute: は「英字」「数字」「-(ダッシュ)」「_
    .....(下線)」のみ有効です',
    .....'alpha_num'=>':attribute: は「英字」「数字」のみ有効です',
    .....'array'=>':attribute: は配列タイプのみ有効です',
    .....'before'=>':attribute: は :date: より前の日付にしてください',
    .....'before_or_equal'=>':attribute: は :date: 以前の日付にしてください',
    .....'between'=>[
    .....'numeric'=>':attribute: は :min: ~ :max: までの数値まで有効です',
    .....'file'=>':attribute: は :min: ~ :max: キロバイトまで有効です',
```

3-4-2.使用言語設定の変更

config\app.phpのlocaleの設定をjaに変更する。

【config\app.php】

```
/*
|-----
| Application Locale Configuration
|-----
|
| The application locale determines the default locale that will be used
| by the translation service provider. You are free to set this value
| to any of the locales which will be supported by the application.
|
*/

- 'locale' => 'en',
+ 'locale' => 'ja',
```

ここで動作確認を行えば、2-4 の動作確認と同じ状態になっている。

3-4-3.フィールド名の日本語化

言語ファイルを編集し、バリデーションメッセージの:attribute部分を日本語化する。
以下のように入力項目のフィールド名にそれぞれ対応した日本語を設定する。

【resources\lang\ja\validation.php】

```
/*
|-----
| Custom Validation Attributes
| カスタム検証属性
|-----
|
| The following language lines are used to swap attribute place-holders
| with something more reader friendly such as E-Mail Address instead
| of "email". This simply helps us make messages a little cleaner.
|
| 次の言語行は、属性プレースホルダを「email」ではなく「E-Mail Address」などの
| 読みやすいものと交換するために使用されます。
|
*/

- 'attributes' => [],
+ 'attributes' => [
+     'name' => '名前',
+     'password' => 'パスワード',
+     'authority' => '権限'
+ ],
```

3-4-4.動作確認

何も入力せず「ログイン」ボタンを押下し、バリデーションエラーメッセージがすべて日本語化されていることを確認する。



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/login'. The page features the HTML5 logo at the top. Below the logo, a red box contains three validation error messages in Japanese: '名前 は必須です' (Name is required), 'パスワード は必須です' (Password is required), and '権限 は必須です' (Permission is required). The form fields are labeled '名前' (Name), 'パスワード' (Password), and '権限' (Permission). The '名前' field has a red border and a red error message '名前 は必須です' below it. The 'パスワード' field has a red border and a red error message 'パスワード は必須です' below it. The '権限' field has two radio buttons, '管理者' (Administrator) and '一般' (General), with the '管理者' button selected. Below the radio buttons is a red error message '権限 は必須です'. At the bottom left, there is a blue button labeled 'ログイン' (Login).

localhost:8000/login

HTML5

- 名前 は必須です
- パスワード は必須です
- 権限 は必須です

名前

パスワード

権限

☒ 管理者 ☐ 一般

ログイン

入力値の保持

バリデーションエラーで戻されるたびに入力をし直すことはユーザにとって非情に面倒である。今回のような入力項目が少ない場合であればまだしも、EC サイトのユーザ登録のような大量の入力項目が存在するページの場合、登録自体が億劫になり、そこで離脱してしまう可能性もある。そういうのはよくない。

Laravel において、直前のリクエストの入力データはフラッシュデータとして保持している。

値を取り出すには、`Request` インスタンスに対し `old` メソッドを使用する。

```
$sample_code = $request->old('fieldName');
```

ビュー側で取り出したい場合は、`old` ヘルパを使用し、以下のように記述する。

```
<input type="text" name="fieldName" value="{{ old('fieldName') }}" />
```

上記の処理をログイントップページに実装する。

今回はコントローラ側で直前の入力値を参照することはないため、ビューのみ変更する。

ただし、パスワードについては再入力させるフォームがほとんどのため、処理は追加しない。

【resources\views\login\login.blade.php】

(略)

```
<label>名前</label>
<input type="text" name="name"
-     class="form-control @if(!empty($errors->first('name'))border-danger
@endif">
+     class="form-control @if(!empty($errors->first('name'))border-danger
@endif"
+     value="{{ old('name') }}">
```

(略)

```
<label>権限</label>
<div class="form-group
@if(!empty($errors->first('authority'))text-danger @endif">
  <div class="radio-inline">
    <label>
-      <input type="radio" name="authority" value="1">管理者
+      <input type="radio" name="authority" value="1"
+      @if (old('authority') == 1) checked @endif>管理者
    </label>
  </div>
  <div class="radio-inline">
    <label>
-      <input type="radio" name="authority" value="2">一般
+      <input type="radio" name="authority" value="2"
+      @if (old('authority') == 2) checked @endif>一般
    </label>
  </div>
  <span class="help-block">{{ $errors->first('authority') }}</span>
</div>
```

(略)

■ 動作確認

以下の項目について動作確認を行う。

1. 名前、パスワードを入力し「ログイン」ボタンを押下。名前の入力保持、パスワードの入力は保持されず、権限未入力のバリデーションエラーが出ることを確認する（参考：画像 1）
2. 名前、権限を入力し「ログイン」ボタンを押下。名前の入力保持、権限の入力保持、パスワード未入力のバリデーションエラーが出ることを確認する（参考：画像 2）

画像 1



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/login'. The page features the HTML5 logo at the top. Below the logo, a red error message box contains the text '権限 は必須です' (Authority is required). The form includes input fields for '名前' (Name) with the placeholder 'なまえ', 'パスワード' (Password), and '権限' (Authority). Under the '権限' label, there are two radio buttons: '管理者' (Administrator) and '一般' (General). Below these radio buttons, the same error message '権限 は必須です' is displayed. At the bottom of the form is a blue 'ログイン' (Login) button.

画像 2

localhost:8000/login

localhost:8000/login シークレット

HTML5

• パスワードは必須です

名前

パスワード

パスワードは必須です

権限

☒ 管理者 ☐ 一般

ログイン

最終段階のソース

フォームリクエストを生成する方法のソースコードのみ示す。

■ resources\views\login\login.blade.php

長いため 2 ページに分割して示す。

```
@extends('layout/layout') @section('content')
<form method="post" action="/login">
  <h2>ログイン</h2>
  @if ($errors->any())
    <div class="alert alert-danger">
      <ul>
        @foreach ($errors->all() as $error)
          <li>{{ $error }}</li>
        @endforeach
      </ul>
    </div>
  @endif {{ csrf_field() }}
</form>
```



```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
use Validator;
use App\Http\Requests\LoginRequest;

class LoginController extends Controller
{
    public function getIndex()
    {
        (略)
    }

    public function postIndex(LoginRequest $request)
    {
        // リクエストパラメータを配列として全件取得
        $input = $request->all();

        // DBのデータと照合
        $db_result = User::where('name', $input['name'])
            ->where('password', $input['password'])
            ->where('authority', $input['authority'])
            ->get();

        // 一致するデータなし
        if (count($db_result) == 0) {
            return view('login/login');
        }

        return view('login/result')->with('input', $input);
    }
}
```

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class LoginRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => 'required',
            'password' => 'required',
            'authority' => 'required|integer|min:1|max:2',
        ];
    }
}
```

```
(略)
/*
|-----
--
| Application Locale Configuration
|-----
--
|
| The application locale determines the default locale that will be used
| by the translation service provider. You are free to set this value
| to any of the locales which will be supported by the application.
|
*/

'locale' => 'ja',

(略)
```

```
(略)
/*
|-----
----
| Custom Validation Attributes
| カスタム検証属性
|-----
----
|
| The following language lines are used to swap attribute place-holders
| with something more reader friendly such as E-Mail Address instead
| of "email". This simply helps us make messages a little cleaner.
|
| 次の言語行は、属性プレースホルダを「email」ではなく「E-Mail Address」などの
| 読みやすいものと交換するために使用されます。
|
*/

'attributes' => [
    'name' => '名前',
    'password' => 'パスワード',
    'authority' => '権限'
],
(略)
```