サーバーサイドスクリプトⅡ

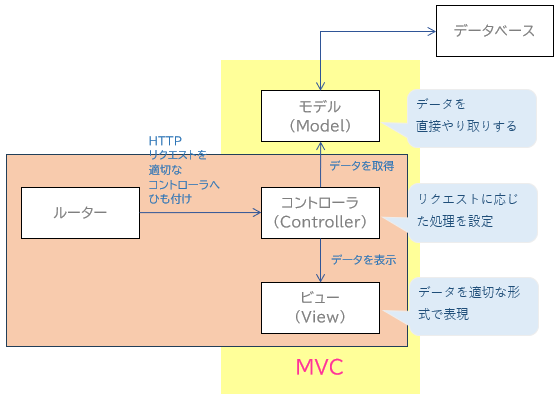
２．Bladeテンプレート

２－１． Blade

1. マイグレーション
2. モデルの作成
3. **コントローラの作成**
4. **ルーティングの設定**
5. **ビューの作成**

今回は、前回のコントローラとルーティングに加えて、ビューを使用します。手順では③～⑤の部分に当たります。

■図：　今回の範囲（赤枠部分）　MVCのVC



２－１－１． Bladeとは

LaravelではBladeというテンプレートエンジンを採用しています。テンプレートエンジンとはHTMLを穴空きにして、穴の部分をプログラムによって動的に出力する仕組みです。Bladeはテンプレート内でプレーンなPHPコードの使用を制限しません。

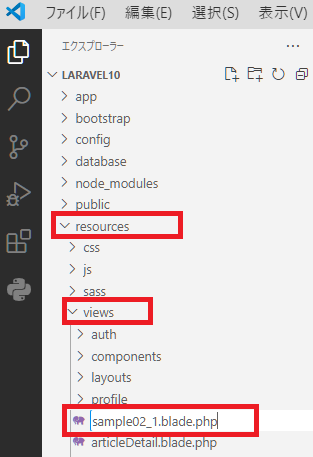
■Bladeファイル作成について

　コマンドではなく、新規ファイル作成にて作成しますが、作成場所と拡張子に注意が必要です。

|  |  |
| --- | --- |
| 作成場所 | [laravelプロジェクトフォルダ]\**resources\views\** |
| 拡張子 | [ファイル名]**.blade.php** |

今回は例として、「sample02\_1.blade.php」ファイルを作成します。

■sample02\_1.blade.php　作成



「sample02.blade.php」ファイル

を新規作成

ファイルが作成できたら、下記のHTMLコードを記載します。

まずは、静的なHTMLのビューを作成します。

■sample02\_1.blade.php コード

<!DOCTYPE html>

<html lang="ja">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Sample02\_1</title>

</head>

<body>

    <h1>サンプル２－１</h1>

</body>

</html>

２－１－２．　viewヘルパ

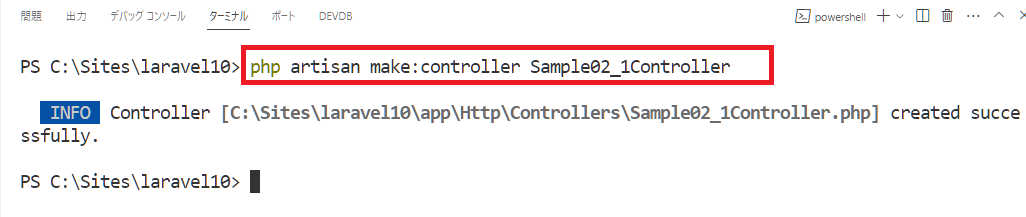
作成した「sample02\_1.blade.php」を表示します。ビューを表示する処理を担うのはコントローラです。

今回の例題用に、Sample02\_1Controllerを作成します。VSCodeの[ターミナル]からコマンドにて、コントローラを作成しましょう。

■Sample02\_1Controller 作成コマンド

　赤枠部分のコマンドを実行する。

※成功すると、ファイル作成場所のパスとともに、「～～successfully」と表示されます。

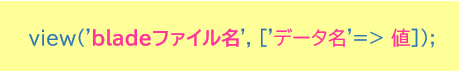


Sample02\_1Controllerに下記のメソッドを追加します。

　■仕様

|  |  |
| --- | --- |
| アクセス修飾子 | public |
| メソッド名 | index |
| 引数 | なし |
| 処理 | sample02\_1.blade.php を呼び出す。  ※呼び出しには、viewヘルパーを使用する（後述） |

■viewヘルパ



■viewヘルパ　引数

|  |  |
| --- | --- |
| ｂladeファイル名 | 「resources\views」配下に作成した拡張子が「.blade.php」のファイル名を指定 |
| [‘データ名’ => 値] | 変数の値などをbladeに渡す場合に指定。（省略可能）  例：変数$testの値をdata1という名前で渡す。  view(‘sample02\_2’, [‘data1’=> $test]) |

今回は、bladeには動的に変更する部分を作成していないので、コントローラから渡すデータはありません。Bladeの呼び出しのみを行います。

　■Sample02\_1Controller.php　indexメソッド

    public function index()

    {

        return view('sample02\_1');

    }

コントローラからviewを呼び出す設定が完了したので、最後にルーティングの設定を行います。

ブラウザから、sample02\_1にGETリクエストがあったとき、Sample02\_1Controllerのindexメソッドが呼び出されるように設定します。

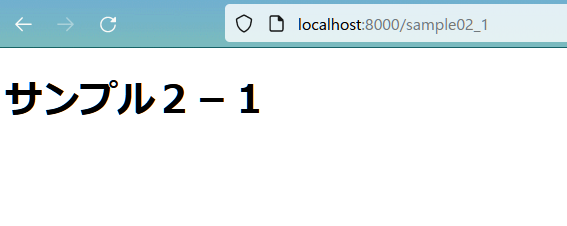
■web.php ルーティングの追加

|  |  |
| --- | --- |
| リスエスト形式 | Get |
| URI | https://localhost:8000**/sample02\_1** |
| 呼び出しクラス・メソッド | Sample02\_1Controller の　index　メソッド  ※use文も忘れずに。 |

記述が完了したら、開発サーバーを起動し、ブラウザで実行結果を確認してください。

Bladeに記述したHTMLの内容が表示されています。

　■実行結果



２－１－３．　変数の値を渡す①

動的に表示を切り替えるために、Bladeに変数を組み込みます。Sample02\_1.blade.phpに下記を追加します。

■sample02\_1.blade.php コード

　※赤枠部分を追加。<h2>タグで囲み、変数$dataの値を組み込み。

<!DOCTYPE html>

<html lang="ja">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Sample02\_1</title>

</head>

<body>

    <h1>サンプル２－１</h1>

<h2>{{ $data }}</h2>

</body>

</html>

　■Bladeでのデータの表示



　　※「 {{ }} 」波括弧で変数を囲む。

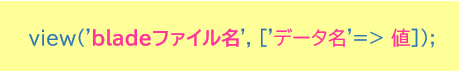
　　「 {{ }} 」で囲むことにより、XSS対策として、htmlspecialchar関数を通してデータをHTMLエンティティへエンコーディングします。

データをエスケープしたくない場合は、「{{!! !!}}」で囲みます。

　■データをエスケープしたくない場合



続いてコントローラから、Bladeへ値を渡します。



今回はBladeに組み込んだ変数が「data」なので、データ名は「data」とします。値には予め用意した変数$tmpの値を渡します。

　■Kadai02\_1Controller.php　indexメソッド

    public function index()

    {

// 変数$tmpを追加

        $tmp = "コントローラから渡した値です。";

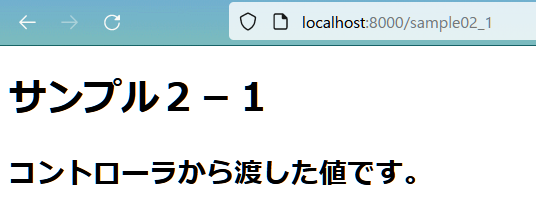
        // viewヘルパの第2引数に値を渡す。

        return view('sample02\_1', ['data' => $tmp]);

    }

今回はアクセスURLにも呼び出すコントローラやメソッドにも変更はないので、ルーティング設定の変更はありません。書き換えが終わったら、ブラウザで表示します。コントローラから渡したデータ表示されていることを確認してください。

　■実行結果



２－１－４．　変数の値を渡す②

viewヘルパの第2引数に複数の値を設定したい場合、compact関数を使用すると見易くなります。

Bladeに3つの変数の値が表示できるように編集してみます。

■sample02\_1.blade.php コード

　※赤枠部分を追加。<h2>タグで囲み、変数$data2と$data3の値を埋め込み。

<!DOCTYPE html>

<html lang="ja">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>Sample02\_1</title>

</head>

<body>

    <h1>サンプル２－１</h1>

<h2>{{ $data }}</h2>

<h2>{{ $data2 }}</h2>

<h2>{{ $data3 }}</h2>

</body>

</html>

続いてコントローラから、Bladeへ値を渡します。

■compact関数



**※変数名とその値から配列を作成します。**

変数名がキー、変数の値がそのキーに関する値となるため、viewでの変数名とコントローラでの変数名を違うものにしたい場合は、compact関数は適しません。

　■Kadai02\_1Controller.php　indexメソッド

    public function index()

    {

        // 変数$tmpを追加 今回は使わないのでコメントアウト

        // $tmp = "コントローラから渡した値です。";

        // 変数 $data1,$data2,$data3 を新規追加

        $data = "コントローラから渡した値です。";

        $data2 = "コントローラから渡した２つめの値です。";

        $data3 = "コントローラから渡した３つ目の値です。";

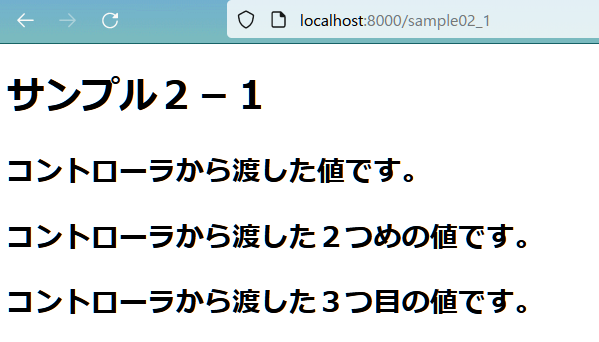
        return view('sample02\_1', compact('data', 'data2', 'data3'));

        // return view('sample02\_1');

    }

今回はアクセスURLにも呼び出すコントローラやメソッドにも変更はないので、ルーティング設定の変更はありません。書き換えが終わったら、ブラウザで表示します。コントローラから渡したデータ表示されていることを確認してください。

　■実行結果



２－２．　Bladeディレクティブ

ディレクティブとは、Bladeで使用することができる、「＠」から始まる範囲内で特定の処理を実行することができる記述方法です。

■主要なディレクティブ一覧

|  |  |
| --- | --- |
| ディレクティブ | 説明 |
| @if(条件)　～　@endif | 条件がtrueのとき、@if ～ @endifまでの内容を出力する |
| @unless(条件) ～ @endunless | 条件がfalseのとき、@unless ～ @endunlessまでの内容を出力する |
| @while(条件) ～ @endwhile | 条件がtrueの間、@while ～ @endwhileの内容を繰り返し出力する |
| @foreach($collection as $element) ～ @endforeach | $collectionの要素を1つずつ$elementに代入しながら、@foreach ～ @endforeachまでの内容を繰り返し出力する |
| @continue | 繰り返し構造の途中で、強制的に次の繰り返しを実行する |
| @break | 繰り返し構造の途中で、強制的に繰り返しを終了する |
| @auth ～ @endauth | 認証済みのとき、@auth ～ @endauth までの内容を出力する |
| @guest ～ @endguest | 未認証のとき、@guest ～ @endguest までの内容を出力する |
| @error(キー名) ～ @enderror | キー名のフィールドにエラーがあるとき、@error ～ @enderrorまでを出力する |
| @csrf | CSRFトークンを生成し、hidden属性で出力する |

「@foreach」を使用するとき、Loop変数を使用できます。繰り返しのインデックスの参照や、繰り返しの最初と最後の判定などが可能です。

■Loop変数

|  |  |
| --- | --- |
| プロパティ | 説明 |
| $loop->index | 繰り返しのインデックスを参照する |
| $loop->iteration | 繰り返しの回数を参照する |
| $loop->first | 最初の繰り返しかどうかを参照する |
| $loop->last | 最後の繰り返しかどうかを参照する |
| $loop->even | 奇数回目の繰り返しかどうかを参照する |
| $loop->odd | 偶数回目の繰り返しかどうかを参照する |
| $loop->parent | @foreachがネストされているとき、親の繰り返しのLoop変数を参照する |

２－２－１．　条件分岐

コントローラから受け取った変数の値に応じて、Bladeでの出力を変更してみます。

コントローラから受け取った変数$numの値が奇数であれば「奇数です」と出力し、そうでなければ「偶数です」と出力します。

Bladeファイル（sample02\_2.blade.php）を作成し、条件分岐を記載します。

■sample02\_2.blade.php コード

　※<body>部分の中のみ記載。

    @if ($num % 2)

        <h3>奇数です</h3>

    @else

        <h3>偶数です</h3>

    @endif

続いてコントローラから、Bladeへ値を渡します。

Sample02\_2Controllerを新規作成し、下記のメソッドを追加します。

　■仕様

|  |  |
| --- | --- |
| アクセス修飾子 | public |
| メソッド名 | index |
| 引数 | なし |
| 処理 | sample02\_2.blade.php を呼び出す。  ※呼び出しには、viewヘルパを使用し、変数$numの値を第2引数に指定する。 |

■Kadai02\_2Controller.php　indexメソッド

    public function index()

    {

        $num = 1;

        return view('sample02\_2', ['num' => $num]);

    }

ビューとコントローラの設定が完了したら、ルーティングの設定を行います。

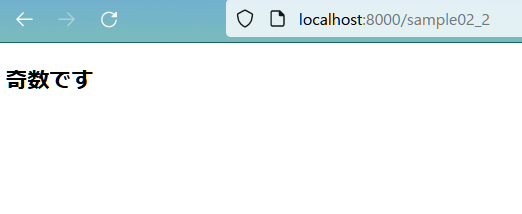
■web.php ルーティングの追加

|  |  |
| --- | --- |
| リスエスト形式 | Get |
| URI | https://localhost:8000**/sample02\_2** |
| 呼び出しクラス・メソッド | Sample02\_2Controller の　index　メソッド  ※use文も忘れずに。 |

記述が完了したら、ブラウザで実行結果を確認してください。

Bladeに記述したHTMLの内容が表示されています。

　■実行結果



コントローラで「$num = 1」としているため、実行結果には「奇数です」と表示されていることを確認してください。正しく表示されていたら、次はコントローラから渡す変数$numの値を変更してみます。

■Kadai02\_2Controller.php　indexメソッド ※$num の値を変更。

    public function index()

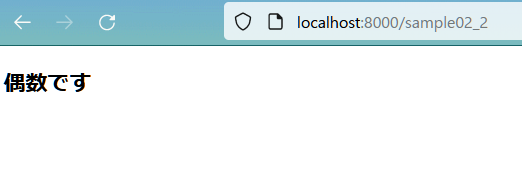
    {

        $num = 10;　// 偶数を設定

        return view('sample02\_2', ['num' => $num]);

    }

■実行結果　※$num の値を偶数にした場合、条件分岐で正しく表示されていること。



２－２－２．　繰り返し

Bladeのディレクティブの記述例を記載します。（Blade部分の例のみ）

　■for

    @for ($i = 0; $i < 10; $i++)

        <p>変数iの値： {{ $i }}</p>

    @endfor

■foreach

    {{-- パターン１（$usersが配列） --}}

    @foreach ($users as $user)

        <p>{{ $user }}</p>

    @endforeach

    {{-- パターン２ --}}

    {{-- ネストされた要素がオブジェクトの場合はアロー演算子を使います --}}

    @foreach ($users as $user)

        <p>{{ $user->name }}</p>

    @endforeach

■while　（変数$numは1以上の整数）

    @while ($num > 0)

        <p>numの値：{{ $num }}</p>

        @php

            $num--;

        @endphp

    @endwhile

２－３． レイアウトの共通化

デザインの共通部分をテンプレートとして作成しておき、各ページにレイアウトを継承して使用することができます。継承にはディレクティブを使用します。

２－３－１．　レイアウトの定義

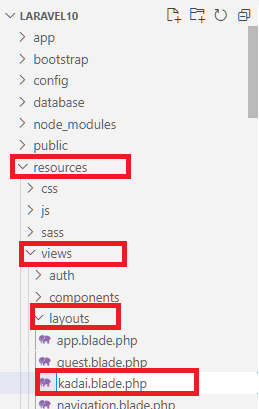
共通部分のレイアウトファイルを作成します。

通常のビューと同様に、Bladeテンプレートを使用しますが、作成場所に注意が必要です。

|  |  |
| --- | --- |
| 作成場所 | [laravelプロジェクトフォルダ]\**resources\views\layouts**  ※viewフォルダ直下ではなく、viewの下の**「layouts」フォルダ下に作成**します。 |
| 拡張子 | [ファイル名]**.blade.php** |

今後の課題で共通レイアウトとして使用する「kadai.blade.php」として作成します。

■kadai.blade.phpファイルの作成



「kadai.blade.php」ファイル

を新規作成

レイアウトファイルを作成したら、以下のコードを記述します。

■kadai.blade.php コード　※赤枠部分が注目キーワードです。説明は後述。

<!DOCTYPE html>

<html lang="ja">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <meta http-equiv="X-UA-Compatible" content="ie=edge">

    <title>@yield('pageTitle') - サーバーサイドスクリプト演習２</title>

    <!-- CSS,JS -->

    @vite(['resources/sass/app.scss', 'resources/js/app.js'])

</head>

<body>

    <div class="wrapper w-screen bg-gray-50">

        <header class="bg-sky-400">

            <div class="container mx-auto py-5">

                <h1 class="text-l text-white mb-6">サーバーサイドスクリプト演習２</h1>

                <h2 class="text-white text-3xl">@yield('title')</h2>

            </div>

        </header>

        <main>

            <div class="container w-full mx-auto py-10">

                @yield('content')

            </div><!--/.container-->

        </main>

    </div><!--/.wrapper-->

</body>

</html>

■@yield（イールド）ディレクティブ

　ページごとに内容が変わるページタイトルやメインコンテンツに使います。主に親テンプレートで使用します。



例題（sample02\_3.blade.php）での使用箇所は3か所です。

|  |  |
| --- | --- |
| @yield(‘pageTitle’) | HTMLのヘッダ部分。<title>タグの内容の一部。 |
| @yield(‘title’) | HTMLのBODY部分。<h2>の内容。タイトルに相当。 |
| @yield(‘content’) | HTMLのBODY部分。メインコンテンツに相当。 |

２－３－２．　レイアウトの拡張

kadai.blade.phpのレイアウトを継承した「sample02\_3.blade.php」ファイルを作成します。

この時、レイアウトを継承する子ファイルは、「resources\views」フォルダ配下に作成することに注意してください。　※layoutsフォルダ配下に作成しない。

■@extendsディレクティブ

　継承には「@extends」ディレクティブを使用します。



sample02\_3.blade.phpへkadai.blade.phpレイアウトを継承してみましょう。

■sample02\_3.blade.php コード

@extends('layouts.kadai')

**※kadai.blade.phpファイルは「layouts」フォルダ配下にあるので、「フォルダ名.ファイル名」となる点に注意すること。**

続いて、テンプレートを継承したページビュー（sample02\_3.blade.php）側から、データを当てはめます。この時、親であるレイアウトテンプレート（kadai.blade.php）に@yieldで指定したセクション名と合わせる必要があります。

■@sectionディレクティブ



「pageTitle」と「title」にそれぞれ文字列を当てはめると、下記のような書き方になります。

■sample02\_3.blade.php コード

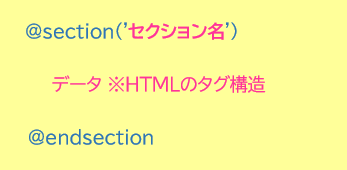
@extends('layouts.kadai')

@section('pageTitle', 'sample02\_3')

@section('title', 'Bladeテンプレート')

メインコンテンツ「content」部分については、複数行に渡るHTMLを記載したいため、下記の記述方法となります。

■@sectionディレクティブ（HTMLデータ複数行）



■sample02\_3.blade.php コード ※赤枠部分を追加

@extends('layouts.kadai')

@section('pageTitle', 'sample02\_3')

@section('title', 'Bladeテンプレート')

@section('content')

    <div>

        <h3 class="text-3xl font-bold py-5 mb-5 border-b-2 border-black">ページビューで指定のh3タグ</h3>

        <div class="p-5">

            <h4 class="text-xl font-bold text-pink-600 mb-2">

                ページビューで指定のh4タグ

            </h4>

        </div>

    </div>

@endsection

ビューが完成したら、**「ターミナル」で「npm run build」コマンドを実行**し、ビルドを行ってください。CSSやJSが反映されます。

■npm run build



ビューが完成したら、続いてコントローラとルーティングの設定を行います。

Sample02\_3Controllerを新規作成し、下記のメソッドを追加します。

　■仕様

|  |  |
| --- | --- |
| アクセス修飾子 | public |
| メソッド名 | index |
| 引数 | なし |
| 処理 | sample02\_3.blade.php を呼び出す。  ※呼び出しには、viewヘルパを使用。 |

■Kadai02\_３Controller.php　indexメソッド

    public function index()

    {

        return view('sample02\_3');

    }

■web.php ルーティングの追加

|  |  |
| --- | --- |
| リスエスト形式 | Get |
| URI | https://localhost:8000**/sample02\_3** |
| 呼び出しクラス・メソッド | Sample02\_3Controller の　index　メソッド  ※use文も忘れずに。 |

記述が完了したら、ブラウザで実行結果を確認してください。

レイアウトが継承されたBladeに、記述したHTMLの内容が表示されています。

　■実行結果

