データベースの処理 - 参照

EloquentORM

Laravelには、データベースとのやり取りをサポートするオブジェクトリレーショナルマッピング（ORM）であるEloquentが含まれています。

Eloquentを使用する場合は、各テーブルに対応する「モデル」が必要になります。

Eloquentモデルでは、テーブルからレコードを取得するだけでなく、テーブルへのレコード挿入、更新、削除も可能です。

オブジェクトリレーショナルマッピング(Object Relational DataBase mapping)とは

オブジェクト指向言語では互いに関連するデータ項目を一つのオブジェクトにまとめ、データを操作する手続き（メソッド）と一体的に管理します。

一方、リレーショナルデータベースでは一件のデータを複数の属性の値の組として表現し、組を連ねた表の形でデータを永続的に保存します。

プログラムからリレーショナルDBへデータを保存するには、オブジェクトを実体化したインスタンスの持つ値をデータベースのテーブル内の項目に当てはめて書き込む操作を行ないますが、通常はデータベース管理システムへ操作を依頼するSQL文などを生成・発行するコードをプログラム中にその都度実装しなければいけません。

オブジェクトリレーショナルマッピングは、この処理を専門的に受け持つもので、あらかじめ設定された対応関係についての情報に基づいて、インスタンスのデータを対応するテーブルへ書き出したり、データベースからデータを読み込んでインスタンスに代入したりといった操作を自動的に行なってくれます。

データベースの準備

ターミナルでmysqlへログインして、授業用のデータベースを作成する用意をしましょう。

データベースの作成

```
-- データベース名：laravel8
CREATE DATABASE `laravel8` DEFAULT CHARACTER SET utf8mb4
```

データベースの確認

```
-- データベースの確認
SHOW DATABASES;
```

マイグレーションの作成と設定

マイグレーションはデータベースのバージョン管理のようなもので、チームがアプリケーションのデータベーススキーマを定義および共有できるようにします。

Schemaファサードは、Laravelがサポートするすべてのデータベースシステムに対し、テーブルを作成、操作するために特定のデータベースに依存しないサポートを提供してくれます。

マイグレーションの作成

マイグレーションの作成は、artisanコマンドの「 **make:migration** 」を使っておこないます。

マイグレーションファイルは、「 **database/migrations** 」の中にLaravelが順序を決定できるようにファイル名にタイムスタンプが付加された状態で作成されます。

```
php artisan make:migration create_{table_name}_table
```

マイグレーションの設定

作成した移行クラスには、「 **up** 」と「 **down** 」の2つのメソッドが用意されます。

upメソッドは、データベースに新しいテーブル、カラム、またはインデックスを追加するために使用し、downメソッドは、upメソッドによって実行する操作を逆にし、以前の状態へ戻す必要があります。

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateArticlesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('articles');
    }
}
```

テーブル構造は、Schemaクラスの「 **create** 」メソッドで定義します。

createメソッドの第一引数はテーブル名を指定し、第２引数は、新しいテーブルを定義するために使用できるBlueprintオブジェクトを受け取るクロージャとなり、Blueprintオブジェクトのクロージャー内で、テーブルのカラム指定をおこないます。

マイグレーションの実行

マイグレーションの実行には、artisanコマンドの「 **migrate** 」でおこないます。

```
php artisan migrate
```

maigrateコマンドには、「 **migrate:rollback** 」「 **migrate:reset** 」「 **migrate:reflesh** 」「 **migrate:fresh** 」などもあり、状況にあわせてデータベースの状態を巻き戻しや作り直しをおこなうことができます。

Laravel 8ドキュメント - マイグレーション
https://readouble.com/laravel/8.x/ja/migrations.ht

シーダーの作成と設定

Laravelは、シードクラスを使用してデータベースにデータをシード（種をまく、初期値の設定）する機能を持っています。

デフォルトで、DatabaseSeederクラスが定義されています。このクラスから、callメソッドを使用して他のシードクラスを実行し、シードの順序を制御することができます。

シーダーの作成

シードファイルの作成は、artisanコマンドの「 **make:seeder** 」を使っておこないます。

作成したシードファイルは、「 **databases/seeders** 」の中に保存されます。

```
php artisan make:seeder {table_name}TableSeeder
```

シーダークラスには、デフォルトで、「 **run** 」メソッドのみ存在します。

このメソッドは、 artisanコマンドの「 **db:seed** 」が実行されるときに呼び出されるため、データベースに挿入するデータを定義できます。

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class ArticlesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        //
    }
}
```

トランザクション

手動でコミットとロールバック

```
try {
    DB::beginTransaction();
    Article::create([
        'title' => 'Laravel DB Transaction',
        'body'  => 'hogehoge',
    ]);

    DB::commit();

} catch (Throwable $e) {
    DB::rollBack();
}
```

自動でコミットとロールバック

例外やエラーがなければコミットし、あればロールバックされます。

```
DB::transaction(function() {
    Article::create([
        'title' => 'Laravel DB Transaction',
        'body'  => 'hogehoge',
    ]);
});

// クロージャー内で変数などを使う場合は、useで指定する
DB::transaction(function() use($title, $body) {
    Article::create([
        'title' => $title,
        'body'  => $body,
    ]);
});
```

シーダーの登録

DatabaseSeederクラス内で、「 **call** 」メソッドを使用して追加のシードクラスを実行できます。
callメソッドを使用することで、データベースのシードを複数のファイルに分割して、単一のシーダークラスが大きくなりすぎないようにできます。

```
<?php

namespace Database\Seeders;

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        // \App\Models\User::factory(10)->create();
        $this -> call([
            ArticlesTableSeeder::class,
        ]);
```

```
    }
}
```

シーダーの実行

artisanコマンドの「 **db:seed** 」で、作成・登録したシーダークラスを実行することができます。

```
php artisan db:seed
```

モデルの作成と設定

授業で使用するテーブル「 articles 」に対応するモデルを作成します。

artisanコマンドで、モデルを作成しましょう。

```
php artisan make:model Article
```

Laravelは、テーブル名は複数形・対応するモデルのクラス名は単数形など、いくつかの命名規則を持っています。

その命名規則を守ることで、自動的に関連付けなどを行ってくれるようになります。

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Article extends Model
{
    use HasFactory;
    use SoftDeletes;    // 論理削除の場合

    protected $table = '{ table_name }';  // テーブル名とモデル名の命名規則を守っている場合は、テーブル名の指定は省略できます。
}
```

コントローラーの作成

データベースが関連するWebアプリケーションの場合は、CRUD（create / read / update / delete）を実装することが大半で、対応したメソッドも数が多くなってしまします。

「 --resource 」を指定してコントローラーを作成すると、自動的にCRUDに対応した各メソッドをクラス内に準備してくれます。

手動で各メソッドを記述しても大丈夫ですが、制作効率を上げてくれます。

```
php artisan make:controller Sample06Controller --resource
```

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class Sample06Controller extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
```

```
    public function destroy($id)
    {
        //
    }
}
```

データベースクエリの実行

データの全件取得

```
$articles = Article::get();
```

Articleモデルを使用するため、使用するモデルをuseで指定する必要があります。

ルーティング

メソッド名の箇所を「 resource 」にすることで、CRUDに対応したルーティング設定を自動的に作成してくれます。

```
Route::resource( 'sample06_1', Sample06_1Controller::class );
```

ルーティングリストの確認

```
php artisan route:list --path=sample06
```