Публикация на тему

Решение проблемы "толстых" контроллеров и моделей архитектурного шаблона проектирования HMVC в фрэймворке Laravel

Проблема толстых контроллеров и моделей архитектурного шаблона проектирования HMVC заключается в том, что если основную логику приложения держать в контроллерах либо моделей, то усложнение самого приложения приводит к утолщению контроллеров либо моделей. Соответственно, чем сложнее взаимодейтсвие контроллеров и моделей, тем становятся толще файлы классов контроллеров либо моделей. Облегчение контроллеров и моделей осуществляется путем введения промежуточных классов и вспомогательных файлов.

Автор
Михалькевич Александр Викторович

Публикация

Наименование Решение проблемы "толстых" контроллеров и моделей архитектурного шаблона проектирования HMVC в фрэймворке Laravel

Автор А.В.Михалькевич

Специальность Проблема толстых контроллеров и моделей архитектурного шаблона проектирования HMVC заключается в том, что если основную логику приложения держать в контроллерах либо моделей, то усложнение самого приложения приводит к утолщению контроллеров либо моделей. Соответственно, чем сложнее взаимодейтсвие контроллеров и моделей, тем становятся толще файлы классов контроллеров либо моделей. Облегчение контроллеров и моделей осуществляется путем введения промежуточных классов и вспомогательных файлов.,

Анотация

Anotation in English

Ключевые слова

Количество символов 105127

Содержание

Введение

1 Описание проблемы

HMVC - основной архитектурный шаблон проектирования, используемый web-разработчиками. HMVC прекрасно справляется с возложенными на него задачами, а именно - распределение логики приложения между контроллерами, моделями и элементами представлений.

Однако, в сложных web-приложениях происходит утолщение контроллеров либо моделей. Причем утолщаются либо классы файлов моделей, если эта логика связана с запросами в базу данных, либо классы файлов контроллеров, если это любая другая серверная логика. Со временем файлы контроллеров или моделей становятся сложными и проблемными в использовании.

2 HMVC для разработки web-приложений

Концепция MVC (Model-View-Controller: модель-вид-контроллер) очень часто используется в разработке web-приложений.

MVC — это архитектурный шаблон проектирования, который описывает способ построения структуры нашего приложения, сферы ответственности и взаимодействие каждой из частей в данной структуре.

Согласно принципам MVC, структура приложения разделяется на три основных компонента, каждый из которых отвечает за различные задачи. Это контроллеры, модели и элементы представления (вид).

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов, нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Модель - это данные и правила, которые используются для работы с данными, которые представляют концепцию управления приложением. В любом приложении вся структура моделируется как данные, которые обрабатываются определённым образом. Что такое

пользователь для приложения — сообщение или книга? Только данные, которые должны быть обработаны в соответствии с правилами (дата не может указывать в будущее, e-mail должен быть в определённом формате, имя не может быть длиннее X символов, и так далее). Модель даёт контроллеру представление данных, которые запросил пользователь (сообщение, страницу книги, фотоальбом, и тому подобное). Модель данных будет одинаковой, вне зависимости от того, как мы хотим представлять их пользователю. Поэтому мы выбираем любой доступный вид для отображения данных. Модель содержит наиболее важную часть логики нашего приложения, логики, которая решает задачу, с которой мы имеем дело (форум, магазин, банк, и тому подобное). Контроллер содержит в основном организационную логику для самого приложения (очень похоже на ведение домашнего хозяйства).

Вид обеспечивает различные способы представления данных, которые получены из модели. Он может быть шаблоном, который заполняется данными. Может быть несколько различных видов, и контроллер выбирает, какой подходит наилучшим образом для текущей ситуации.

Веб приложение обычно состоит из набора контроллеров, моделей и видов. Контроллер может быть устроен как основной, который получает все запросы и вызывает другие контроллеры для выполнения действий в зависимости от ситуации.

HMVC - это эволюция концепции MVC, которая используется в многих веб приложениях. Она появилась как решение некоторых проблем, проявившихся при использовании MVC в веб приложениях. Решение было представлено на сайте JavaWorld в июле 2000. Предлагалось использовать стандартную триаду Модель-Контроллер-Вид использовать в качестве слоев в «**иерархии родитель-потомок**». Рисунок отображает принцип работы:

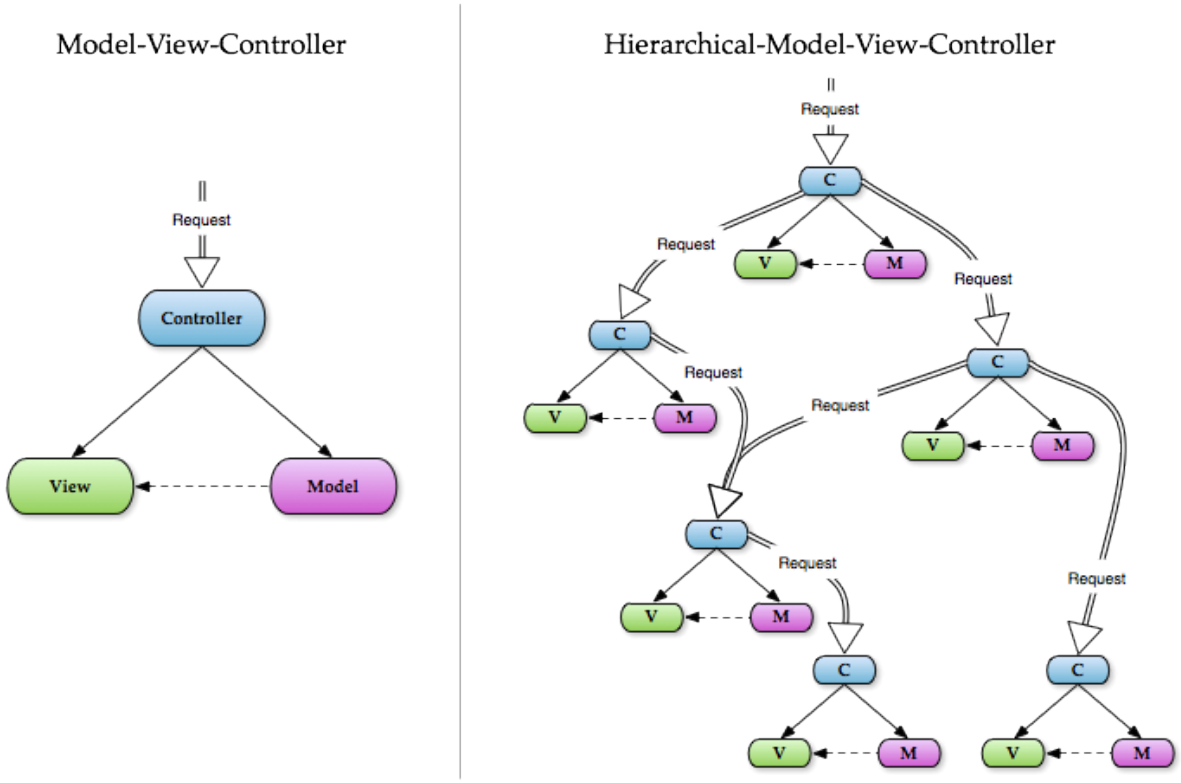

Рисунок 1 – Принцип работы HMVC

Каждая триада функционирует независимо от других. Триада может запросить доступ к другой триаде через их контроллеры. Обе точки позволяют приложению распределяться по

нескольким местам, если нужно. В дополнение, использование слоев триад MVC позволяет добиться более глубокой и тщательной разработки приложений. Такой подход позволяет получить ряд преимуществ, которые будут описаны далее по тексту. Почему следует использовать HMVC? Ключевыми преимуществами, которые дает использование HMVC при разработке приложения, являются:

Модульность: Уменьшается зависимость между различными частями приложения.

Организация: Наличие папки для каждой значимой триады облегчает работу по загрузке приложения на сервер сайта.

Повторное использование: В следствии природы дизайна приложения, очень просто повторно использовать практически каждый кусок кода.

Расширяемость: Делает приложение доступным для расширения без жертвования легкостью поддержки.

Данные преимущества позволяют делать более совершенные приложения при снижении сложности разработки.

Маршрутизация (англ. Routing) — процесс определения маршрута следования данных в сетях связи. Маршруты могут задаваться административно (статические маршруты), либо вычисляться с помощью алгоритмов маршрутизации, базируясь на информации о топологии и состоянии сети, полученной с помощью протоколов маршрутизации (динамические маршруты).

Статическими маршрутами могут быть:

маршруты, не изменяющиеся во времени;

маршруты, изменяющиеся по расписанию;

Маршрутизация в компьютерных сетях выполняется специальными программно-аппаратными средствами — маршрутизаторами; в простых конфигурациях может выполняться и компьютерами общего назначения, соответственно настроенными.

3 Усовершенствование HMVC в фрэймворке Laravel

Усовершенствование концепции HMVC в фрэймворке Laravel заключается во введении вспомгательных и промежуточных решений применяемых во взаимодействии контроллеров с моделями. Такими решениями являются:

middleware, или подготовительное программное обеспечение,
serviceProvider, или классы библиотек,
другие решения.

Middleware

HTTP Middleware (подготовительное программное обеспечение) - это фильтры обработки HTTP-запроса. Так, например, в Laravel включены middlewares для проверки аутентификации пользователя. Если пользователь не авторизирован, middleware перенаправляет его на страницу логина. Если же авторизирован - middleware не вмешивается в прохождение запроса, передавая его дальше по цепочке middleware-посредников к собственно приложению.

Middlewares можно использовать в качестве фильтров для роутов.

Конечно, проверка авторизации - не единственная задача, которую способны выполнять middlewares. Это также добавление особых заголовков (например, CORShttp-ответ вашего приложения) или логирование всех http-запросов.

В Laravel есть несколько дефолтных middleware, которые находятся в папке app/Http/Middleware. Это middlewares для реализации режима обслуживания сайта ("сайт временно не работает, зайдите позже"), проверки авторизации, CSRF-защиты и т.п.

Создание middleware

Для создания middleware можно воспользоваться встроенной artisan-командой make:middleware:

```
php artisan make:middleware NameMiddleware
```

ServiceProvider

Основная задача поставщика услуг, или ServiceProviders – это загрузка вспомогательных, в том числе, и собственных классов.

Если открыть конфигурационный файла app.php, то мы увидим там множество провайдеров, определенных в массиве providers. Это все классы, которые будут загружены для приложения. Многие из них являются отложенными, что означает, что они не будут загружаться при каждом запросе, а только тогда, когда они действительно необходимы, т.е. когда они будут вызываться.

Все провайдеры должны наследоваться от класса Illuminate\Support\ServiceProvider. Данный абстрактный класс требует, чтобы был определен хотя бы один метод класса-провайдера: register. В этом методе регистрируется вспомогательный класс.

Сам провайдер можно создать с помощью artisan:

```
php artisan make:provider NameServiceProvider
```

Так выглядит пустой класс ServiceProvider

```
class SizeServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap the application services.
     *
     * @return void
     */
    public function boot()
    {
    }
```

```
    /**
     * Register the application services.
     *
     * @return void
     */
    public function register()
    {
    }
}
```

Пространство имени App\Providers, в котором находится этот класс сервис-провайдера, - место для хранения сервис-провайдеров нашего Laravel-приложения, но мы можете располагать свои сервис-провайдеры где угодно внутри PSR-4 папки (если вы не меняли composer.json, то это папка app).

После вызова методов register() всех сервис-провайдеров приложения, вызывается метод boot(). Там уже можно использовать весь существующий функционал классов фреймворка и приложения - регистрировать слушателей событий, подключать роуты и т.п.

Метод register() предназначен для загрузки вспомогательных классов с предопределенными входящими параметрами, которые определяются в этом же методе. Для всего остального предназначен метод boot().

Другие решения

Использование в разработке middleware и serviceProvider-ов помогают облечить контроллеры. Однако это не все усовершенствования, привнесенные в HMVC фрэймворком Laravel. Стоит еще отметить классы Request и прослушиватели моделей. Но это уже темы отдельных публикаций.

Заключение

Список использованных источников

Приложения