laravel 要php開放擴展

extension openssl

extension PDO

extension Mbstring

extension Tokenizer

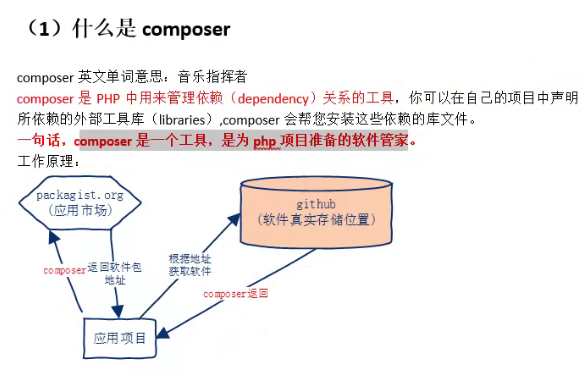


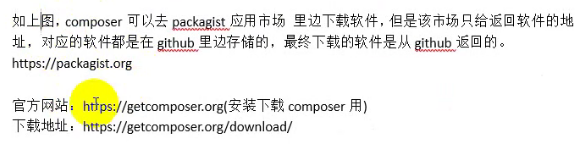
驗證碼的代碼依賴需要fileinfo.dll

curl 用於請求發送(傳api)

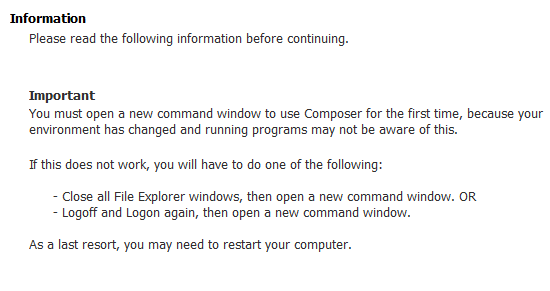
apache24/conf/httpd.conf 也要開放2個擴展。

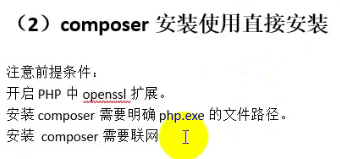
composer 介紹





composer安裝





在cmd下直接命令 composer 就會出現version info。

開始使用composer

第一步, 中國鏡像命令(我們跳過)

第二步, composer部署laravel項目

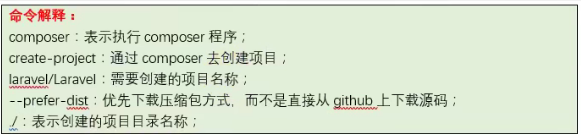
比如創建一個命為laravel的laravel項目

在想要創建laravel的cmd路徑下使用命令 (D:\webdocs)

2種方法:

composer create-project --prefer-dist laravel/laravel ./${project.name}

laravel new ${project.name}



默認是創建最新version的laravel, 如果想要指定version: (5.4.\*的版本)

composer create-project --prefer-dist laravel/laravel=5.4.\* ./project.name

更新composer 命令

#composer self-update

(了解) laravel直接安裝 (不用composer) 開頭1分鐘(不推薦使用)

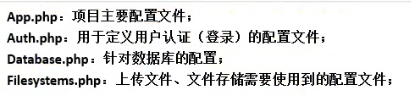
<https://www.bilibili.com/video/BV1V4411h7JE?p=3>

目錄結構分析



技巧：app 目錄中的許多類可以由 Artisan 通過命令生成。要查看可用的命令，請在終端中運行 php artisan list make 命令。

config目錄



database目錄



migrations內的是用於 創建數據表的類文件

public目錄



重點: 項目單一入口文件也放在public下。

因此, 後續配置 虛似主機時需要將站點位置指定到public下。

resources目錄



resources 內lang與views



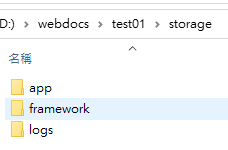
\*\*\*

routes目錄

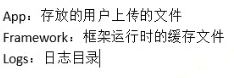
定義路由的目錄, web.php是定義路由的文件

storage目錄

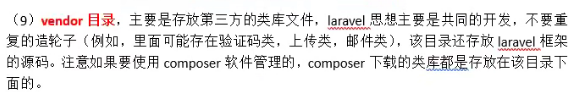




app , framework , logs



vendor 目錄



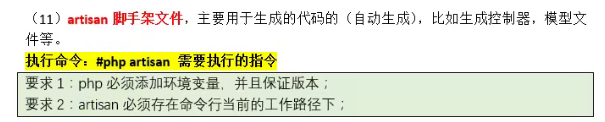
.env文件



.env是config配置文件, 比如要改mysql的連接認證。

artisan 腳手架文件, 用於自動生成代碼

命令: php artisan 需要執行的指令



composer.json 依賴包配置文件 (類似lock-package.json)



重點需要常用的目錄



啟動laravel 框架

方式1

cmd在project的路徑下: 輸入指令 php artisan serve

(這種啟動方式與apache httpd.exe無關)

不推薦使用: 因為這種方式只能跑php, 不能跑database。

方式2

使用wamp或lamp環境

虛擬主機配置 (虛擬主機 =/= 虛擬機)

\*\*\*

修改apache的虛擬主機

C:\Apache24\conf\extra\httpd-vhosts.conf 文件

#列表顯示 (開發環境) Options Indexes FollowSymLinks

開發時可以開啟列表, 上線時就不要開了。

Options –indexes 就是關。

<VirtualHost 127.65.43.21:80>

#配置站點管理員email, 當站點產生500錯誤時,會顯示在頁面上

ServerAdmin catcat1997@windowslive.com

# 增加站點(根目錄)

DocumentRoot 'D:/webdocs/test01/public'

# 增加主機名 (綁定域名)

ServerName www.laravel80.com

# 服務器域名的別名

ServerAlias laravel80.com

# 配置權限

<Directory 'D:/webdocs/test01/public'>

#設定方式: 代表匹配權限順序

#Order Deny,Allow

#限定範圍: 按照順序執行

#Deny from all

#Allow from all

# Order 2.4Apache寫法

Require all granted

#列表顯示 (開發環境)

Options Indexes FollowSymLinks

#指定默認訪問文件

DirectoryIndex index.html index.htm index.php

</Directory>

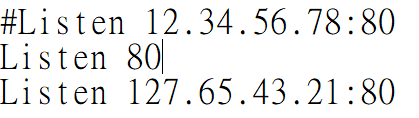
</VirtualHost>

C:\Apache24\conf 內

httpd.conf

原本Listen 80 要加上 Listen 127.65.43.21:80 (我們用80 port, 但不同ip地址)

如果用8080port 輸入domain時就要再加上:8080, 十分不美和不友好。



修改host文件(線上叫DNS域名解析)

C:\Windows\System32\drivers\etc\hosts

由於windows的hosts文件不能指定port, 所以如果我們訪問domain([www.laravel8080.com)時](http://www.laravel8080.com)時), 不想輸入port號(laravel8080.com:8080)。

就只好用不同的ip地址。 127.65.43.21



\*\*\* 因為瀏覽器默認80port是主要的, 不用特別輸入。

windows的hosts文件也是默認去80 端口號。

總結:

httpd.conf下

Listen 127.65.43.21:80

httpd.vhosts.conf下

<VirtualHost 127.65.43.21:80>

window的hosts下

127.65.43.21 www.laravel80.com laravel80.com

我們想手機也能訪問到 (電腦路由ip 192.168.0.116)

由於手機沒有listen 127.65.43.21 所以不可能訪問到,

所以我們再多開一個本地電腦的port放lavarel80.com, 我們放到\*:8080port

這樣192.168.0.116:8080 就能訪問到我們的test01 project項目了。

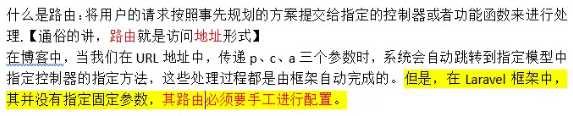
加入

<VirtualHost \*:8080>

Listen 8080

laravel 路由的使用規則

路由定義



路由文件位置 routes/web.php

使用routes前 apache的httpd-vhosts文件 要加上AllowOverride All\*\*\*

<Directory 'D:/webdocs/test01/public'>

#設定方式: 代表匹配權限順序

#Order Deny,Allow

#限定範圍: 按照順序執行

#Deny from all

#Allow from all

# Order 2.4Apache寫法

AllowOverride All

Require all granted

Options Indexes FollowSymLinks

</Directory>

說明

AllowOverride引數就是指明Apache伺服器是否去找.htacess檔案作為配置檔案，如果設定為none,那麼伺服器將忽略.htacess檔案，如果設定為All,那麼所有在.htaccess檔案裡有的指令都將被重寫。對於AllowOverride，還可以對它指定如下一些能被重寫的指令型別.

通常利用Apache的rewrite模組對 URL 進行重寫的時候， rewrite規則會寫在 .htaccess 檔案裡。但要使 apache 能夠正常的讀取.htaccess 檔案的內容，就必須對.htaccess 所在目錄進行配置。從安全性考慮，根目錄的AllowOverride屬性一般都配置成不允許任何Override ，即

AllowOverride none

.htaccess是在Apache服務器下的一個對於系統目錄進行各種權限規則設置的文件，存在於Linux操作系統中。  
簡單來說，**重新定向，啟用壓縮，導向URL，利用瀏覽器快取和更多的事情，都可以透過這個小文件來完成。**

.htaccess is in your application file /public。 laravel項目中public文件中。

要開啟apache 的 mod\_rewirte才能使用。

uri 與 url 的分別 url uri

URI（Uniform Resource Identifier）統一資源識別符號

URL（Uniform Resource Locatin）統一資源定位符

URL 是 URI 的子集

URL 是路徑，URI 是資源實際位置。

URI：localhost:80/user/save

URL：localhost:80/user

很清晰的就能看出要想找到 URI，必先找到它的 URL，這也就解釋了 URL 是 URI 的子集是啥意思了。只找到 URL 是不知道要做什麼的，只有找到 URI 才知道資源是啥，要幹些什麼事情。

路由寫法

Route::請求方法(相對路徑,callback);

view(path); view方法, 是laravel的渲染方法

Route::get($uri, $callback);

Route::post($uri, $callback);

Route::put($uri, $callback);

Route::patch($uri, $callback);

Route::delete($uri, $callback);

Route::options($uri, $callback);

\*\*\*

Route::match

Route::any

有的時候你可能需要注冊一個可響應多個 HTTP 請求的路由，這時你可以使用 match 方法，也可以使用 any 方法注冊一個實現響應所有 HTTP 請求的路由：

Route::match(['get', 'post'], '/', function () {

//

});

Route::any('/', function () {

//

});

CSRF保護

Laravel 可以輕松使地保護你的應用程序免受 跨站請求偽造（CSRF）攻擊。 跨站點請求偽造是一種惡意攻擊，它憑借已通過身份驗證的用戶身份來運行未經過授權的命令。

Laravel 會自動為每個活躍的用戶的會話生成一個 CSRF「令牌」。該令牌用於驗證經過身份驗證的用戶是否是向應用程序發出請求的用戶。

無論何時，當您在應用程序中定義 HTML 表單時，都應該在表單中包含一個隱藏的 CSRF 標記字段，以便 CSRF 保護中間件可以驗證該請求，你可以使用 @csrf Blade 指令來生成令牌字段，如下：

<form method="POST" action="/profile">

@csrf

...

</form>

包含在 web 中間件組裡 VerifyCsrfToken [中間件](https://learnku.com/docs/laravel/8.x/middleware)會自動驗證請求裡的令牌是否與存儲在會話中令牌匹配

路由參數

Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {

//

});

路由的參數通常都會被放在 {} 內，並且參數名只能為字母，而不能包含 - 符號。如果有需要，可以用下劃線 (\_) 代替 - 。路由參數會按路由定義的順序依次注入到路由回調或者控制器中，而不受回調或者控制器的參數名稱的影響。

可選參數

有時，你可能需要指定一個路由參數，但你希望這個參數是可選的。你可以在參數後面加上？標記來實現，但前提是要確保路由的相應變量有默認值：

Route::get('user/{name?}', function ($name = null) {

return $name;

});

Route::get('user/{name?}', function ($name = 'John') {

return $name;

});

注意: 路由參數與 回調函數的參數 順序要一致。

比如:

Route::any("/user/{name}/said/{comment?}",function($name,$comment = 'hi'){

    echo "user's name: " .$name . " said: " . $comment;

});

路由參數與 回調函數的參數,同時

{name} 與$name放第一。

{comment}與$comment放第二。

### 正則表達式約束

你可以使用路由實例上的 where 方法約束路由參數的格式。where 方法接受參數名稱和定義參數應如何約束的正則表達式：

Route::get('user/{id}/{name}', function ($id, $name) {

//

})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);

php中

引用一個類的屬性和方法就使用->符號。

路由命名

路由命名可以方便地為指定路由生成 URL 或者重定向。

*// 路由命名 ->name('路由命名');*

Route::get('user/profile', function () {

    echo 'user profile page';

})->name('profile');

你還可以指定控制器行為的路由名稱：

Route::get('user/profile', [UserProfileController::class, 'show'])->name('profile');

通過在路由定義上鏈式調用 name 方法可以指定路由名稱

注意：路由命名必須是唯一的

生成指定路由的 URL

一旦為路由指定了名稱，就可以使用全局輔助函數 route 來生成鏈接或者重定向到該路由。

// 生成鏈接...

$url = route('profile');

// 生成重定向...

return redirect()->route('profile');

\*\*\*

如果有定義參數的命名路由，可以把參數作為 route 函數的第二個參數傳入，指定的參數將會自動插入到 URL 中對應的位置：

name()

route()

Route::get('/user/profile/{page}', function ($page) {

    echo 'user profile page' . $page;

})->name('user\_profile');

redirect()

Route::get('/redirect',function(){

    return redirect()->route('user\_profile',['page' => 100]);

});

如果在數組中傳遞額外的參數，這些鍵或值將自動添加到生成的 URL 的查詢字符串中：

Route::get('/redirect2',function(){

    return redirect()->route('user\_profile',['page' => 100 , 'photo' => 'yes']);

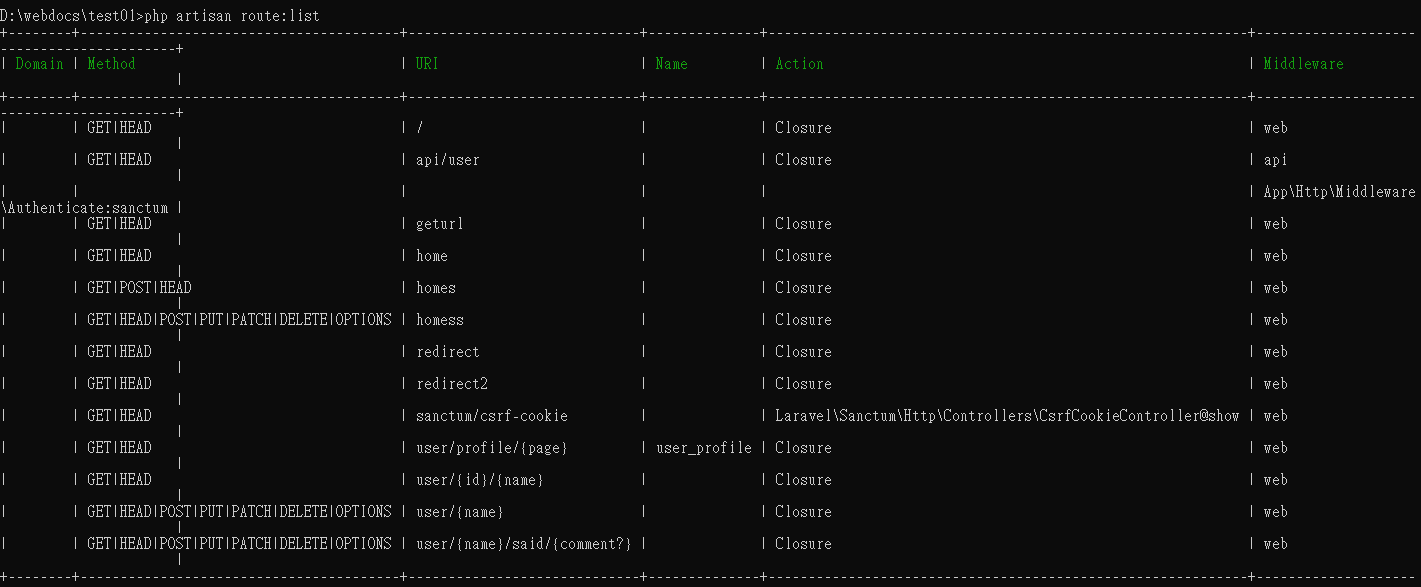
});

*// http://laravel80.com/user/profile/100?photo=yes*

技巧：有時候，你可能希望為某些 URL 參數的請求范圍指定默認值，例如在本地環境，你可以使用 [URL::defaults 方法](https://learnku.com/docs/laravel/8.x/urls#default-values)。

<https://learnku.com/docs/laravel/8.x/urls/9372#default-values>

本看系統存在的路由的命令: php artisan route:list (laravel項目路徑下)



api/user 是調用中間件的位置

檢查當前路由(了解)

如果你想判斷當前請求是否指向了某個命名過的路由，你可以調用路由實例上的 named 方法。例如，你可以在路由中間件中檢查當前路由名稱：

/\*\*

\* 處理一個請求

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @return mixed

\*/

public function handle($request, Closure $next)

{

if ($request->route()->named('profile')) {

//

}

return $next($request);

}

路由群組

### 路由前綴

prefix 方法將會為路由組中的每一個 URI 添加前綴。例如，您可以給該組中所有的 URI 添加 admin 的前綴：prefix() group()

*// 路由前綴 路由群組*

Route::prefix('/admin')->group(function(){

    Route::get('/',function(){ echo 'main admin page';});

    Route::get('/users',function(){ echo 'admin/users page';});

    Route::get('/players',function(){ echo 'admin/players page';});

});

*// 路由名稱前綴*

*// name 方法可以為路由組中每一個路由名稱添加一個指定的字符串作為前綴。例如，*

*// 您可以給已經分組的路徑添加 admin 的前綴。給定的字符串與指定的路由名稱前綴完全相同，*

*// 因此我們將確保在前綴中提 供尾部的 . 字符：*

*// http://laravel80.com/user  路由名是 admin.user*

Route::name('admin.')->group(function () {

    Route::get('user', function () {

*// Route assigned name "admin.users"...*

        echo 'admin user page';

    })->name('user');

});

控制器

為了替代在路由文件中以閉包形式定義的所有的請求處理邏輯，你可能想要使用控制類來組織這些行為。控制器能將相關的請求處理邏輯組成一個單獨的類。例如，一個 UserController 類可以處理所有與用戶相關的傳入請求，包括顯示，創建，更新和刪除用戶，控制器被存放在 app/Http/Controllers 目錄。

編寫控制器

基礎控制器

命名: 自定義名+Controller.php

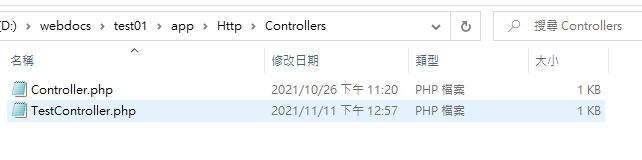
eg: MyControlController.php

可用命令 創建控制器

php artisan make:controller 名Controller

php artisan make:controller TestController



.

你可以通過 Artisan 命令工具裡的 make:controller 命令中的 --invokable 選項來生成可調用的控制器：

php artisan make:controller ProvisionServer --invokable

ProvisionSerevr是controller的名

PHP 命名空間(namespace)

PHP 命名空間可以解決以下兩類問題：

1. 用戶編寫的代碼與PHP內部的類/函數/常量或第三方類/函數/常量之間的名字沖突。
2. 為很長的標識符名稱(通常是為了緩解第一類問題而定義的)創建一個別名（或簡短）的名稱，提高源代碼的可讀性。

## use 用於命名空間的別名引用

這個想必在日常的工程化開發中會非常常見。畢竟現在的框架都是使用了命名空間的，不管做什麼都離不開各種類依賴的調用，在各種控制器文件的上方都會有大量的use xxx\xxx\xxx;語句。

namespace 命名空間三元素: 常量,方法,類

class TestController extends Controller

由於我們生成的TestController與Controller.php在同一目錄, 所以不用引入也可以extends繼承。

use 也可以用來引入。

### 單行為控制器

如果控制器操作特別復雜，你可能會發現將整個控制器類專用於單個操作會很方便。為此，你可以在控制器內定義一個單一的 \_\_invoke 方法：

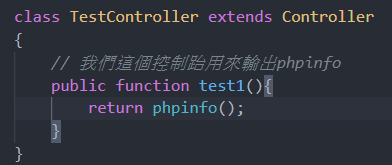
生成單行為控制器

php artisan make:controller Test2Controller --invokable

控制器路由

自定義路由 方法function





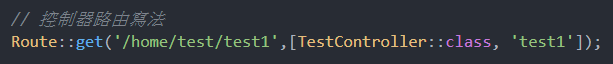
編寫路由規則:

設定路由: /home/test/test1 /p/c/a

路由指定uri調用控制器內的方法



use App\Http\Controllers\TestController;



當一個請求與指定路由的 URI 匹配時，App\Http\Controllers\TestController 控制器中的 test1 方法將會執行。路由參數也將會被傳遞給該方法。

控制器 分目錄



分目錄的創建控制器

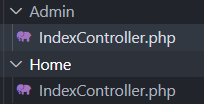
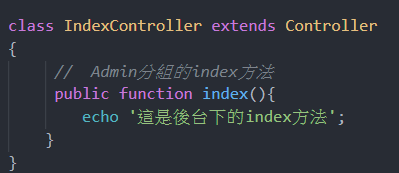
php artisan make:controller Admin/IndexController

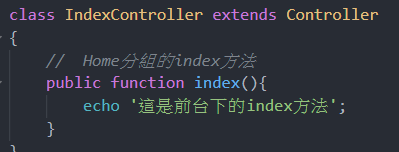
php artisan make:controller Home/IndexController

用命令生成控制器的好處: 會自動幫我們引入Controller (因為與Controller在不同目錄下, 所以要引入才能extends)

use App\Http\Controllers\Controller;

編寫不同分目錄下的控制器:



編寫對應路由

*// 分目錄管理控制器*

use App\Http\Controllers\Home\IndexController as homeIndexController;

Route::get('home/index/index',[homeIndexController::class,'index']);

use App\Http\Controllers\Admin\IndexController as adminIndexController;

Route::get('admin/index/index',[adminIndexController::class,'index']);

*// 另一種書寫路由控制器的方法*

*// 路徑@方法*

Route::get('/admin/index/index', 'App\Http\Controllers\Admin\IndexController@index');

由於[IndexController::class] 是對應於use路徑的最終位置。

如果重名了,就會亂, 所以可以用as。 區分分目錄下同名的控制器。

兩種書寫路由控制器的方法

use App\Http\Controllers\UserController;

Route::get('/users', [UserController::class, 'index']);

// or

Route::get('/users', 'App\Http\Controllers\UserController@index');

*// 另一種書寫路由控制器的方法*

*// 路徑@方法*

Route::get('/admin/index/index', 'App\Http\Controllers\Admin\IndexController@index');

在類的成員方法裡面，可以通過$this->property(property是屬性名字)這種方式來訪問類的屬性、 方法，但是要訪問類的靜態屬性或者在靜態方法裡面卻不能使用，而是使用self::$property。  
  
在類的非靜態方法裡面可以使用偽變量$this，這個偽變量是調用該方法的實例化對象引用（一般來說該對像是這個方法所在類的實例化對象， 但是也有可能是另外的類的對象，比如在另一個類裡面使用靜態化來調用這個類的方法 ）。

\*\*\*

在成員方法中，呼叫成員方法的方法是物件名加方法名，格式就是“物件名->方法名”。但是在定義類的時候，我們往往不知道物件名是什麼，所以就沒法用物件名，這時，我們就要用到偽變數$this。

$this的意思就是本身，在$this中有個指標，誰呼叫它，他就指向誰，它只能再類內部使用。

\*\*\*

$\_GET[‘name’] 可理解為 $\_GET->name (使由於$\_GET不是類,不能用->)

-> 就是js類內的鏈接 person.name (person類內的屬性name)

在php 就是person->name

php -> 符號是“插入式解引用操作符”（infix dereference operator）。換句話說，它是調用由引用傳遞參數的子程序的方法（當然，還有其它的作用）。正如我們上面所提到的，在調用PHP的函數的時候，大部分參數都是通過引用傳遞的。PHP中的->功能就和它們在Perl或C++中一樣。下面是一個簡單的解引用的例子：

$user->friends\_count  
取对象的friends\_count属性。

$t->homeTimeline($p)  
调用对象的homeTimeline方法，方法中传入一个参数$p

    在PHP的腳本中=>操作符時很常見的。因為php數組函數很豐富，我們要經常用到數組，因為它操作數據很方便。

$phparr= new array( in => 'reply,'

side => 'left',

padx => 2m,

pady => 2m,

ipadx => 2m,

ipady => 1m

)

:: 范圍解析操作符（也可稱作 Paamayim Nekudotayim）或者更簡單地說是一對冒號，可以用於訪問靜態成員，類常量，還可以用於覆蓋類中的屬性和方法。

當在類定義之外引用到這些項目時，要使用類名。

namespace 與 use

**為了方便管理類引入了**[**命名空間**](https://link.zhihu.com/?target=https%3A//www.baidu.com/s%3Fwd%3D%25E5%2591%25BD%25E5%2590%258D%25E7%25A9%25BA%25E9%2597%25B4%26tn%3DSE_PcZhidaonwhc_ngpagmjz%26rsv_dl%3Dgh_pc_zhidao)**namespace。**

**簡單的說namespace 就是將裡面的類打個包。**

use Illuminate\Support\Facades\DB;

以上的use 就是引入 namespace Illuminate\Support\Facades下的DB類

DB類寫在Facades文件夾內的DB.php內(因為Facades文件夾內所有php檔都同時命名為namespace Illuminate\Support\Facades 所以我們能找到DB類)。

同時因為是static方法, 引入後不需要實例化也能使用。

namespace**後面跟的是名稱 不是路徑！**

我們是調用DB類, 不是調用DB方法。

DB類內有多種方法 DB::xxx

HTTP Request

請求

Laravel 的 Illuminate\Http\Request 類提供了一種面向對象的方法，可以與應用程序處理的當前 HTTP 請求進行交互，以及檢索與請求一起提交的輸入內容，cookies 和文件。

illuminate 位置 path

vendor\laravel\framework\src

請求路徑和方法

illighte\Http\Request 實例提供了多種方法來檢查傳入的 Http 請求，並擴展了 Symfony\Component\HttpFoundation\Request 類。下面我們將討論幾個最重要的方法。

檢索請求路徑

path 方法返回請求的路徑信息。因此，如果傳入請求的目標是 http://example.com/foo/bar，則 path 方法將返回 foo/bar：

$uri = $request->path();

檢查請求路徑 / 路由

is 方法允許您驗證傳入的請求路徑是否與給定的模式匹配。使用此方法時，可以使用 \* 字符作為通配符：

if ($request->is('admin/\*')) {

//

}

使用 routeIs 方法，可以確定傳入請求是否與[命名路由](https://learnku.com/docs/laravel/8.5/routing#named-routes)匹配：

if ($request->routeIs('admin.\*')) {

//

}

檢索請求 URL

要檢索傳入請求的完整 URL，你可以使用 url 或 fullUrl 方法。

url 方法將返回不包含查詢字符串的 URL，

而 fullUrl 方法包含查詢字符串：

$url = $request->url();

$urlWithQueryString = $request->fullUrl();

檢索請求方法

method 方法將返回請求的 HTTP 動詞。你可以使用 isMethod 方法來驗證 HTTP 動詞是否匹配給定的字符串：

$method = $request->method();

if ($request->isMethod('post')) {

//

}

請求頭

你可以使用 header 方法從 Illuminate\Http\Request 實例中檢索一個請求頭。如果請求中不存在該頭，將返回 null。然而，header 方法接受一個可選的第二個參數，如果請求中不存在該頭，將返回該參數：

$value = $request->header('X-Header-Name');

$value = $request->header('X-Header-Name', 'default');

hasHeader 方法可用來決定請求是否包含一個給定的頭：

if ($request->hasHeader('X-Header-Name')) {

//

}

輸入

檢索所有輸入數據

可以使用 all 方法以 array 的形式檢索所有傳入請求的輸入數據。無論傳入的請求是來自 HTML 表單還是 XHR 請求，都可以使用此方法。

$input = $request->all();

檢索一個輸入值

使用一些簡單的方法，可以從 Illuminate\Http\Request 實例獲取所有的用戶輸入數據，而不用在意用戶使用的是哪種 HTTP 動詞(method I guess)。不管是什麼 HTTP 動詞， input 方法都可以用來獲取用戶的輸入數據：

$name = $request->input('name');

可以將默認值作為第二個參數傳遞給 input 方法。 如果請求中不存在第一個參數指定的字段的輸入值，則將返回此值：

$name = $request->input('name', 'Sally');

\*\*\*

input(鍵,default值) 獲取$\_GET或$\_POST[指定鍵]的值, 如果沒有,可以定default值。 input就是$\_GET與$\_POST的結合。

input()方法的使用 , 不用在意用戶使用的是哪種 HTTP method發送請求。

你可以使用 all 方法來獲取 array 類型的全部輸入數據：

$input = $request->all();

all() 就是$\_POST / $\_GET 但沒有指定想要獲取的鍵的值。

laravel dd()函數 就是dump and die() 停止下面的代碼。 用於開發時測試。

dd($input);

當處理包含數組的表單時，可以使用 「.」 運算符來訪問數組的數據：

$name = $request->input('products.0.name');

$names = $request->input('products.\*.name');

#### 從查詢字符串獲取輸入 query() 方法

input 方法可以從整個請求體中獲取數據（包括查詢字符串）, 而 query 方法僅僅從查詢字符串中獲取輸入值：

$name = $request->query('name');

如果請求中所需要的查詢字符串字段不存在，第二個參數將會被返回：

$name = $request->query('name', 'Helen');

你也可以使用無參數的 query 方法來獲取全部查詢條件的關聯數組：

$query = $request->query();

input all query 都需要 use Illuminate\Http\Request; 來使用

函數的參數也要加上Reuqest $request

通過動態屬性獲取輸入

你也可以通過 Illuminate\Http\Request 接口實例的動態屬性訪問用戶的輸入。例如你的一個表單中包含 name 字段，則可以通過下面這種方式獲取：

$name = $request->name;

獲取 JSON 輸入

當向應用傳遞 JSON 請求時，只要將請求頭中的 Content-Type 設置為 application/json 後你便可以使用 input 方法來獲取 JSON 數據。你也可以使用「.」語法獲取 JSON 數組內容：

$name = $request->input('user.name');

獲取輸入的布爾值

當處理諸如 checkbox 類的 HTML 元素時，你的應用可能獲取到以字符串形式傳遞的 “真假值”，例如 “true” 或者 “on”。為了方便，你可以使用 bool 方法來將這些值轉換布爾類型。boolean 方法將會把 1、"1"、true、"true"、"on" 和 "yes" 返回為 true。而其他值將會返回 false：

$archived = $request->boolean('archived');

#### 獲取部分輸入數據 only() except()

如果需要獲取輸入數據的子集，你可以使用 only 或 except 方法。它們接受單個 array 或者動態參數列表：

$input = $request->only(['username', 'password']);

$input = $request->only('username', 'password');

$input = $request->except(['credit\_card']);

$input = $request->except('credit\_card');

#### 判斷輸入值是否存在

你可以使用 has 來判斷當前請求中是否含有指定的值。如果請求中存在該值則 has 方法將會返回 true：

if ($request->has('name')) {

//

}

當給定一個數組時，has 將會判斷指定的值是否全部存在：

if ($request->has(['name', 'email'])) {

//

}

hasAny 方法將會在指定的值有一個存在的情況下返回 true：

if ($request->hasAny(['name', 'email'])) {

//

}

如果你想要判斷一個值在請求中是否存在，並且不為空，可以使用 filled 方法：

if ($request->filled('name')) {

//

}

如果你想要判斷一個值在請求中是否缺失，可以使用 missing 方法：

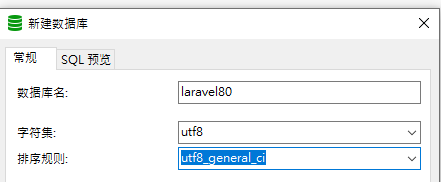
if ($request->missing('name')) {

//

}

DB類操作數據庫

我們用破解的navicat 15



utf8\_general\_ci 是不區分大小寫。

創建數據庫與測試表

create table member(

id int primary key auto\_increment,

name varchar(32) not null,

age tinyint unsigned not null,

email varchar(32) not null

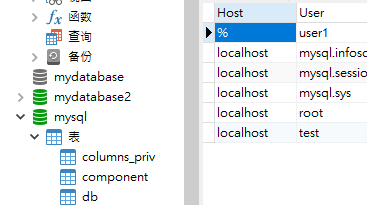
) engine myisam charset utf8;

\*\*\*

數據庫在laravel框架中的配置

在.env文件里面。(在項目根目錄)

在數據庫中 mysql庫中的user表 如果沒有127.0.0.1



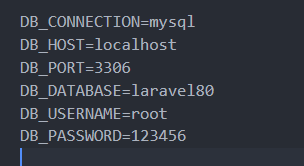
.env 的 DB\_HOST 就不能輸入127.0.0.1了。 (host:% 是任何)

上圖我們只有user1是%

所以我們root的host 只能用localhost



.env中



也可以在config 下 database.php文件里面配置

使用env函數, 表示先由env文件里面獲取, 如果獲取失敗,就使用env函數的第二個參數。



\*\*\*

在Test控制器中引入DB門面。

引入簡寫, 可以在config/app.php內查看。 aliases 就是可簡寫的引入



引入簡寫



增刪改查,需要4個routes。

增 /home/test/add

刪 /home/test/delete

改 /home/test/update

查 /home/test/select

執行原生 SQL 查詢

一旦配置好數據庫連接後，便可以使用 DB facade 門面運行查詢。DB facade 為每種類型的查詢提供了相應的方法：select，update，insert，delete 和 statement。

Controller中

use Illuminate\Support\Facades\DB;

執行 Select 查詢

你可以使用 DB Facade 的 select 方法來運行基礎的查詢語句：

public function index()

{

$users = DB::select('select \* from users where active = ?', [1]);

return view('user.index', ['users' => $users]);

}

傳遞給 select 方法的第一個參數就是一個原生的 SQL 查詢，而第二個參數則是需要綁定到查詢中的參數值。通常，這些值用於約束 where 語句。參數綁定可以防止 SQL 注入。

使用命名綁定 :xx

除了使用 ? 表示參數綁定外，你還可以使用命名綁定的形式來執行一個查詢：

$results = DB::select('select \* from users where id = :id', ['id' => 1]);

執行 Insert 語句

你可以使用 DB Facade 的 insert 方法來執行 insert 語句。與 select 方法一樣，該方法將原生 SQL 查詢作為其第一個參數，並將綁定的數據作為第二個參數：

 DB::insert('insert into member values (?,?,?,?)', [null,'mary',30,'mary123@gmail.com']);

執行 Update 語句

update 方法用於更新數據庫中現有的記錄。該方法返回該執行語句影響的行數：

$affected = DB::update("update member set name = 'new\_mary' where name = ?",['mary']);

執行 Delete 語句

delete 方法用於從數據庫中刪除記錄。與 update 方法一樣，返回受該執行語句影響的行數：

 public function delete($id = 3){

        $deleted = DB::delete('delete from member where id= ?',[$id]);

        echo 'affected' . $deleted . 'row';

    }

執行普通語句

有些數據庫語句不會有任何返回值。對於這些語句，你可以使用 DB Facade 的 statement 方法來運行：

DB::statement('drop table users');

statement 可以執行任何SQL語句, 但沒有返回值。

*// statement 可以執行任何SQL語句, 但沒有返回值。*

        DB::statement('insert into member values (?,?,?,?)', [null,'mary',30,'mary123@gmail.com']);

視圖 views

位置在 resources/views下 在views下是可以進行分目錄管理的。

技巧：如果您想瞭解更多關於如何編寫 Blade 模板的更多信息？查看完整的 [Blade 文檔](https://learnku.com/docs/laravel/8.x/blade) 將是最好的開始

Blade 是 Laravel 提供的一個簡單而又強大的模板引擎。和其他流行的 PHP 模板引擎不同，Blade 並不限制你在視圖中使用原生 PHP 代碼。所有 Blade 視圖文件都將被編譯成原生的 PHP 代碼並緩存起來，除非它被修改，否則不會重新編譯，這就意味著 Blade 基本上不會給你的應用增加任何負擔。Blade 視圖文件使用.blade.php 作為文件擴展名，被存放在 resources/views 目錄。

視圖 views

視圖包含應用程序的 HTML 服務，並且將控制器 / 應用程序邏輯與演示邏輯分開。視圖文件存放於 resources/views 目錄下。一個簡單的視圖代碼如下所示：

<!-- View stored in resources/views/greeting.blade.php -->

<html>

<body>

<h1>Hello, {{ $name }}</h1>

</body>

</html>

將上述代碼存儲到 resources/views/greeting.blade.php 後，我們可以使用全局輔助函數 view 將其返回，例如：

Route::get('/', function () {

return view('greeting', ['name' => 'James']);

});

如你所見， 傳遞給 view 輔助函數的第一個參數對應 resources/views 目錄中視圖文件的名稱。第二個參數是應該可供視圖使用的數據數組。在這種情況下，我們傳遞 name 變量，該變量將使用 Blade syntax 在視圖中顯示。

\*\*\*

當然，視圖文件也可以嵌套在 resources/views 目錄的子目錄中。「點」符號可以用來引用嵌套視圖。例如，如果你的視圖存儲在 resources/views/admin/profile.blade.php，則可以這樣引用它：

return view('admin.profile', $data);

\*\*\*

注意：視圖目錄名中不應該包含 . 字符。

判斷視圖文件是否存在

如果需要判斷視圖文件是否存在，可以使用 View facade。如果存在，exists 方法會返回 true :

use Illuminate\Support\Facades\View;

if (View::exists('emails.customer')) {

//

}

向視圖傳遞數據

正如在前面的示例中所看到的，可以將一組數據傳遞給視圖：

Route::get('/greeting/{name?}',function($name = 'peter'){return view('greeting',['name'=>$name]);});

*// 視圖變量的渲染方法  1.數組, 2.->with(數組) , 3.->with()->with()*

        return view('greeting',['name1'=>$name1 , 'name2'=> $name2]);

        return view('greeting')->with(['name1'=>$name1 , 'name2'=> $name2]);

        return view('greeting')->with('name1','test1')->with('name2','test2');

以這種方式傳遞信息時，數據應該是具有鍵 / 值對的數組。在視圖中，可以使用相應的鍵訪問每個值，例如 <?php echo $key; ?>。作為將完整的數據數組傳遞給 view 輔助函數的替代方法，您可以使用 with 方法將各個數據片段添加到視圖中：

return view('greeting')->with('name', 'Victoria');

與所有視圖共享數據

有時候，你可能需要共享一段數據給應用程序的所有視圖。 你可以在服務提供器的 boot 方法中調用視圖門面（Facade）的 share 方法。例如，可以將它們添加到 AppServiceProvider 或者為它們生成一個單獨的服務提供器：

use Illuminate\Support\Facades\View;

public function boot() {

View::share('key', 'value');

}

php compact函數 (與laravel無關)

用於打包數組。

compact — 建立一个数组，包括变量名和它们的值

**compact** ( [mixed](mk:@MSITStore:D:\PHP_8\manual\php_enhanced_zh.chm::/res/language.types.declarations.html#language.types.declarations.mixed) $var\_name , [mixed](mk:@MSITStore:D:\PHP_8\manual\php_enhanced_zh.chm::/res/language.types.declarations.html#language.types.declarations.mixed) ...$var\_names ) : array

視圖優化

默認情況下，視圖是按需編譯的。當執行渲染視圖的請求時，Laravel 將確定該視圖的編譯版本是否存在。如果已編譯視圖存在，Laravel 將比較未編譯視圖是否已被修改。如果已編譯視圖不存在，或者未編譯視圖已被修改，Laravel 將重新編譯該視圖。

在請求期間編譯視圖會對性能產生影響，因此 Laravel 提供了 view：cache Artisan 命令來預編譯應用中使用的所有視圖文件。為了提高性能，你可能希望在部署過程中運行以下命令：

\*\*\*

php artisan view:cache

你可以使用 view：clear 命令清除視圖緩存：

php artisan view:clear

laravel 時區: 在config/app.php下

'timezone' => 'Asia/Macau'

模板中使用函數

<h1>現在的時間是: {{date('Y-m-d H:i:s',$time)}}</h1>

可以寫php / laravel的函數。

laravel視圖的foreach循環



@foreach @endforeach 循環體 沒有大括號 (用於view.blade.php內渲染)

</head>

<body>

    id&emsp;&emsp;name&emsp;&emsp;age&emsp;&emsp;email <br/>

    @foreach($data as $val)

        {{$val->id}}&emsp;&emsp;{{$val->name}}&emsp;&emsp;{{$val->age}}&emsp;&emsp;{{$val->email}} <br/>

    @endforeach

</body>

</html>

laravel視圖的if else

@if(xxx)

執行語句

@elseif()

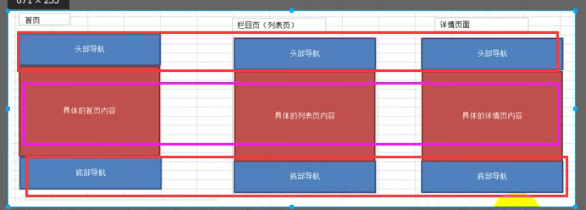
執行語句

@else

執行語句

@endif

模板繼承/包含



案例: 編寫父級頁面, 再寫子類頁面(相當於php 中的兩個類)

\*\*\*

**1. 模板中的@方法:**

@yield是在模板中定義區塊

在extends頁面中, @section與@endsection內寫入模板中@yield的內容。

@section 在模板中定義區塊 如果在模板中用section,就要同時用@show來結尾

@show 是@endsection同時@yield 用於結束@section

**2. extend頁面中的@方法:**

@extends 就會直接繼承指定的整個頁面。

@section @endsection是在模板中的定義區塊@yield代替什麼想寫的內容。

@parent 是在@section中使用, 用來添加@section的內容

(在模板的@section完結後添加)。

*<!-- 文件的包含 include -->*

@include('template.template')

視圖中外部靜態文件引入方式 (了解)

./ 是當前

/ 是根目錄開始

所以給瀏覽器的地址, 以/ 開頭, 就會直接回到根目錄找路徑。

以前的引入方式 (由於沒有對應路徑的路由,所以會返回404 not found)

<link rel="stylesheet" type="text/css" href="/resources/css/app.css">

asset() 方法:

<link rel="stylesheet" type="text/css" href="{{asset('/resources/css/app.css')}}">

[Laravel 項目開發規范](https://learnku.com/docs/laravel-specification/7.x) /

靜態資源

css, js, font, 圖片，用戶頭像等靜態資源 必須 使用 CDN 加載。

用戶上傳的靜態資源 應該 酌情處理。

第三方 CDN

項目中使用了第三方前端類庫中的 CSS/JS/FONTS 等靜態資源，絕不 使用第三方鏈接進行加載（包括第三方 CDN）。

第三方類庫靜態資源 必須 下載並納入代碼版本管理器中。

\*\*\*\*\*

laravel默認只有public下的靜態資源能夠訪問。

要訪問到對應的資源，直接在模板上使用 {{ asset('靜態文件路徑') }}

比如訪問public下的index.php (訪問的根目錄就是public)

<link rel="stylesheet" type="text/css" href="/index.php">

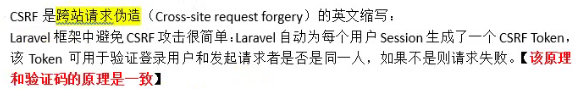
<link rel="stylesheet" type="text/css" href="/robots.txt">

<link rel="stylesheet" type="text/css" href="{{asset('/index.php')}}">

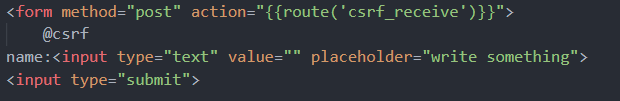
CSRF攻擊

corss-site request forgery

比如, 別人知道我們的post地址, 在自己的網站寫個form表單,但action寫我們的post地址,去改我們的數據。



案例: 創建2個路由, 一個用於展示表單(get), 一個處理請求(post)



有了@csrf , 我們才能成功把表單內容提交到csrf\_receive。

使用 @csrf Blade 指令來生成令牌字段

包含在 web 中間件組裡 VerifyCsrfToken [中間件](https://learnku.com/docs/laravel/8.x/middleware)會自動驗證請求裡的令牌是否與存儲在會話中令牌匹配。

after using @csrf , The csrf token verification is directly involved with your session

\*\*\*在laravel中,csrf機制是自動開啟的, 所以如果沒有用@csrf生成令牌, 就會出現419 page expire 無法提交表單。

Laravel 將當前的 CSRF 令牌存儲在一個 XSRF-TOKEN cookie

中，該 cookie 包含在框架生成的每個響應中。你可以使用 cookie 值來設置 X-XSRF-TOKEN 請求頭。

這個 cookie 主要是作為一種方便的方式發送的，因為一些 JavaScript 框架和庫，例如 Angular 和 Axios ，會自動將它的值放入 X-XSRF-TOKEN 頭中。

@csrf的另一種寫法

{{csrf\_field()}}

也是用在views視圖網頁中的。

@csrf 檢查網頁原始碼, 會生成, @CSRF就是一個TOKEN



但如果是異步提交表單, 還是用@csrf比較好。

csrf 白名單 (排除例外路由)

app\Http\Middleware\VerifyCsrfToken.php 就是白名單的中間件



在$except方法下,可以寫白名單的網站。

protected $except = [

*//*

        '/home/csrf\_receive',

    ];

使csrf\_receive不用@csrf也能夠成功接收到表單內容。

如果 ‘\*’ 就是全部網站都是白名單。(所有路由排除csrf驗證)

模型操作 AR模式

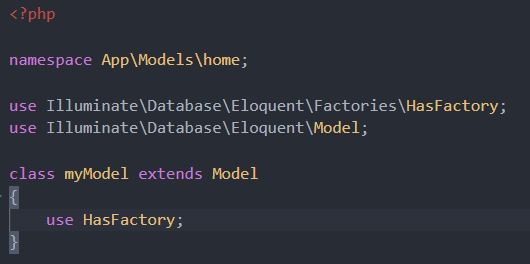
Laravel 的 Eloquent ORM 提供了一個漂亮、簡潔的 ActiveRecord 實現來和數據庫交互。每張數據表都有一個對應的「Model模型」用來與該表交互。你可以通過模型查詢數據表中的數據，以及在數據表中插入新記錄。

ORM 對象操作模型

創建模型最簡單的方法就是使用 make:model [Artisan 命令](https://learnku.com/docs/laravel/8.x/artisan)：

php artisan make:model home/myModel

生成的初始模型代碼



namespace 命名空間一定要放在第一行\*\*\*

數據表名稱 $table

請注意，我們並沒有告訴 Eloquent 我們的 myModel 模型使用哪個數據表。 除非明確地指定了其它名稱，否則將使用類的復數形式「蛇形命名」來作為表名。因此，在這種情況下，Eloquent 將假設 myModel 模型存儲的是 myModel s 數據表中的數據，而 AirTrafficController 模型會將記錄存儲在 air\_traffic\_controllers 表中。

\*\* 所以請一定要指定$table。

protected $table = 'member';

主鍵

Eloquent 也會假設每個數據表都有一個名為 id 的主鍵列。你可以定義一個受保護的 $primaryKey 屬性來重寫約定。

protected $primaryKey = 'id';

此外，Eloquent 假設主鍵是一個自增的整數值，這意味著默認情況下主鍵會自動轉換為 int 類型。如果您希望使用非遞增或非數字的主鍵則需要設置公共的 $incrementing 屬性設置為 false：

 public $incrementing = false;

如果你的主鍵不是一個整數，你需要將模型上受保護的 $keyType 屬性設置為 string：

protected $keyType = 'string';

時間戳

默認情況下，Eloquent 預期你的數據表中存在 created\_at 和 updated\_at 兩個字段 。如果你不想讓 Eloquent 自動管理這兩個列， 請將模型中的 $timestamps 屬性設置為 false：

public $timestamps = false;

如果需要自定義時間戳的格式，在你的模型中設置 $dateFormat 屬性。這個屬性決定日期屬性在數據庫的存儲方式，以及模型序列化為數組或者 JSON 的格式：

protected $dateFormat = 'U';

如果你需要自定義存儲時間戳的字段名，可以在模型中設置 CREATED\_AT 和 UPDATED\_AT 常量的值來實現：

const CREATED\_AT = 'creation\_date';

const UPDATED\_AT = 'last\_update';

數據庫連接

默認情況下，Eloquent 模型將使用你的應用程序配置的默認數據庫連接。如果你想為模型指定一個不同的連接，設置 $connection 屬性即可：

protected $connection = 'connection-name';

默認屬性值

如果要為模型的某些屬性定義默認值，可以在模型上定義 $attributes 屬性：

protected $attributes = [ 'delayed' => false, ];

\*\*\*\*\*

批量賦值

你也可以使用 create 方法來保存新模型。 此方法會返回模型實例。 不過，在使用之前，你需要在模型上指定 fillable 或 guarded 屬性，因為所有的 Eloquent 模型都默認不可進行批量賦值。

當用戶通過請求傳入意外的 HTTP 參數，並且該參數更改了數據庫中你不需要更改的字段時，就會發生批量賦值漏洞。 比如：惡意用戶可能會通過 HTTP 請求傳入 is\_admin 參數，然後將其傳給 create 方法，此操作能讓用戶將自己升級成管理員。

所以，在開始之前，你應該定義好模型上的哪些屬性是可以被批量賦值的。你可以通過模型上的 $fillable 屬性來實現。

protected $fillable = ['id','name','age','email'];

一旦我們設置好了可以批量賦值的屬性，就可以通過 create 方法插入新數據到數據庫中了。 create 方法將返回保存的模型實例：

模型在控制器的引入

模型在控制器中調用

引入member模型類

在myModelController中

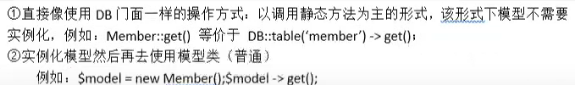
*// 引入myModel模型*

use App\Models\home\myModel;

一旦我們設置好了可以批量賦值的屬性，就可以通過 create 方法插入新數據到數據庫中了。 create 方法將返回保存的模型實例：

$member = App\Models\myModel::create(['name' => 'Flight 10']);

模型使用有2種方式 (下圖為5.x version laravel)



插入及更新模型

插入

要往數據庫新增一條記錄，先創建新模型實例，給實例設置屬性，然後調用 save 方法：

public function myModel\_add(Request $request){

*// 實例化myModel模型*

        $member = new myModel;

        $member->id = $request->id;

        $member->name = $request->name;

        $member->age = $request->age;

        $member->email = $request->email;

        $member->save();

    }

更新

save 方法也可以用來更新數據庫已經存在的模型。更新模型，你需要先檢索出來，設置要更新的屬性，然後調用 save 方法。同樣，updated\_at 時間戳會自動更新，所以也不需要手動賦值：

public function myModel\_update(Request $request){

*// save 方法也可以用來更新數據庫已經存在的模型。更新模型，你需要先檢索出來，設置要更新的屬性，然後調用 save 方法。*

        $member = new myModel;

*// 這裡我們不想寫表單去接收數據, 就直接給$request接收的數據賦值*

        $request->id = 10;

        $request->name = 'mary\_Model';

        $request->age = 99;

        $request->email = 'myModel\_update\_testing@gmail.com';

*// find()方法 找到想update的對應id*

        $member = myModel::find($request->id);

        $member->name = $request->name;

        $member->age = $request->age;

        $member->email = $request->email;

        $member->save();

    }

批量更新

也可以更新匹配查詢條件的多個模型。在這個示例中，所有的 name 等於 mary\_Model 和age為 99 的會update：

myModel::where('name', 'mary\_Model')

          ->where('age', 99)

          ->update(['age' => 80, 'name'=>'mary\_update\_multi']);

\*\*\*

update 方法需要一個鍵為字段名和值為對應數據的數組，該數組代表應更新的字段。

firstOrNew 方法像 firstOrCreate 方法一樣嘗試通過給定的屬性查找數據庫中的記錄。不過，如果 firstOrNew 方法找不到對應的模型，會返回一個新的模型實例。注意 firstOrNew 返回的模型實例尚未保存到數據庫中，你需要手動調用 save 方法來保存

$flight = App\Models\Flight::firstOrCreate(['name' => 'Flight 10']);

updateOrCreate

你還可能遇到希望更新現有模型或在不存在的情況下則創建新的模型的情景。 Laravel 提供 updateOrCreate 方法來一步實現。 類似於 firstOrCreate 方法，updateOrCreate 持久化模型，因此無需調用 save()：

// 如果有從奧克蘭到聖地亞哥的航班，則價格定為 99 美元...

// 如果沒匹配到存在的模型，則創建一個...

$flight = App\Models\Flight::updateOrCreate(

['departure' => 'Oakland', 'destination' => 'San Diego'],

['price' => 99, 'discounted' => 1]

);

刪除模型

可以在模型實例上調用 delete 方法來刪除實例：

通過主鍵刪除模型

在上面的例子中，在調用 delete 之前需要先去數據庫中查找對應的模型。事實上，如果你知道了模型的主鍵，你可以直接使用 destroy 方法來刪除模型，而不用先去數據庫中查找。destroy 方法除了接受單個主鍵作為參數之外，還接受多個主鍵，或者使用數組、集合來保存多個主鍵：

myModel::destroy(1);

myModel::destroy(1, 2, 3);

myModel::destroy([1, 2, 3]);

myModel::destroy(collect([1, 2, 3]));

use Illuminate\Database\Eloquent\Model;

高級子查詢

Selects 子查詢

\*\*\*

使用 get 方法獲取查詢結果：

myModel extends Model方法

->save() 插入目標/更新目標數據

where() 條件查詢

->update() 結合where()使用 用於更新目標數據

create() 新寫一個數據 , 與save()結合用來插入新數據。

find() 根據主鍵查詢

destroy()根據主鍵刪除

->delete()刪除目標

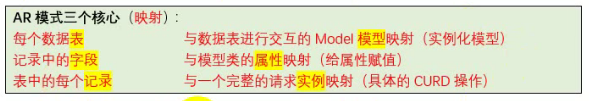
select() select 語句

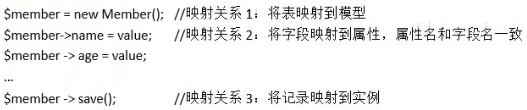
->orderBy() orderBy語句

\*\*\*\*\*

->get() 給變量賦值數據(獲取查詢結果)

$datas = myModel::select('\*')->orderBy('id','asc')->get();





Request $request語法

$request->all()

$request->input

$request->input(‘name’)

$request->only([])

$request->except([])

$request->has([])

$request->filled()

$request->missing()

$request->get()

:: 與 -> 強化理解

myModel::find() 靜態調用類方法

$member = new myModel;

$member->find() 實例化再調用類內方法

create不需要save()就直接插入數據了

myModel::create(['id'=> null,'name'=>'test','age'=>44,'email'=>'testing@test.com']);

find()結合toArray()

$member = myModel::find($request->id)->toArray();

\*\*\*\*\*\*\*

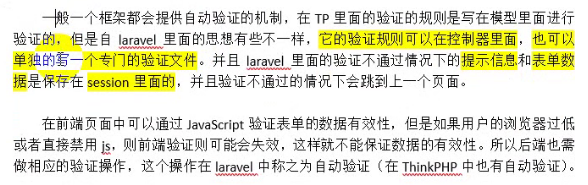
$request 就是$\_POST或$\_GET用來接收數據。

$request->name 等價於 $request->input(‘name’)

通過動態屬性獲取輸入 $request->name

自動驗證(重點)

TP是ThinkPHP



驗證方式: 控制器方式驗證

ValidatesRequests trait



$request->method() 取得頁面的請求方式(返回字符串String)。

ValidatesRequests類的validate方法

Illuminate\Foundation\Validation\ValidatesRequests::validate

[validate](https://laravel.com/api/8.x/Illuminate/Foundation/Validation/ValidatesRequests.html#method_validate)([Request](https://laravel.com/api/8.x/Illuminate/Http/Request.html) $request, array $rules, array $messages = [], array $customAttributes = [])

Validate the given request with the given rules.

Trait 是為類似PHP 的單繼承語言而准備的一種代碼復用機制。 Trait 為了減少單繼承語言的限制，使開發人員能夠自由地在不同層次結構內獨立的類中復用method。 Trait 和Class 組合的語義定義了一種減少復雜性的方式，避免傳統多繼承和Mixin 類相關典型問題。

validate rule api網

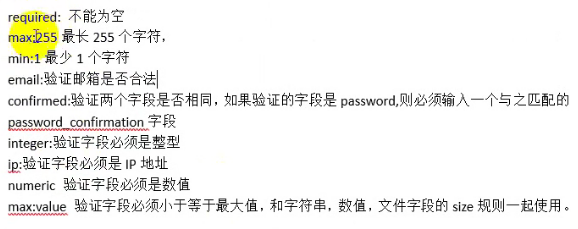
<https://laravel.com/api/8.x/Illuminate/Validation/Rules.html>

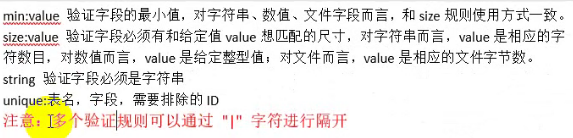
<https://www.tutorialspoint.com/laravel/laravel_validation.htm>

\*\*\*\*\*

<https://laravel.com/docs/8.x/validation#rule-size>

5.x version validate rules

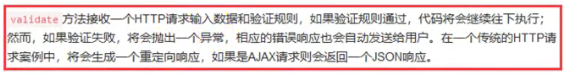




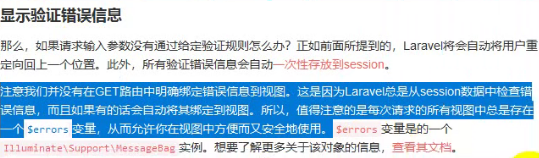
多個驗證規則用 | 分開

$this->validate(數據對象,[驗證規則]);

validatesRequests不需要引入, 因為是基類的。



顯示驗證錯誤信息 $errors



@if(count($errors) > 0)

        <div class="alert alert-danger">

            <ul>

                @foreach($errors->all() as $error)

                    <li>{{$error}}</li>

                @endforeach

            </ul>

        </div>

    @endif

all()方法, 取出全部數據(數組形式)

之後把error信息轉為中文輸出

我們框架同有英文支持

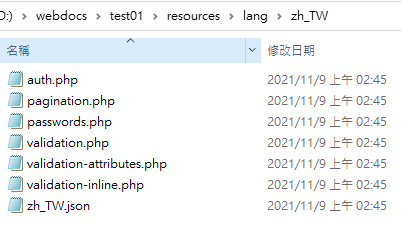
我們需要去下載一個中文支持包。

packagist.org

search: laravel-lang 就下載最多人用的。

我們下載的是caoue/laravel-lang

* Choose the branch:
  + laravel4 for Laravel 4 project
  + master for Laravel 5, 6, 7, 8 projects
* Copy the folders of languages that you want, in resources/lang folder of your Laravel application (or app/lang in Laravel 4).



## Usage 使用laravel - lang

Laravel only

In the file config/app.php, change the value of locale by the short name of your language (app/config/app.php in Laravel 4).

去config/app 配置



改為

zh\_TW



這樣錯誤信息也會變為中文。

因為zh\_TW有validation 文件。

\*\*\*\*

還可以到validation文件中改出錯信息。 改為想要的樣子。

\*\* 為了更好的理解laravel代碼,我們轉回英文好了。

文件上傳



我們想上傳用戶avatar頭像

先在sql數據庫加上avatar字段名

alter table member add column avatar varchar(100) character set utf8 collate utf8\_general\_ci not null after email;

collate是校對集 用utf8\_general\_ci。

上傳文件 表單要加enctype屬性

如果表單form沒有enctype=’ multipart/form-data’, 就沒有$\_FILES變量了。

<form method="POST" action="" enctype="multipart/form-data">

\*\*\*

表單action為空, 就是傳給自己(以post方式)。

之前在控制器中, 添加上傳業務的處理邏輯。(看手冊)

illuminate/Http/Request實例提供file方法或者動態屬性來訪問上傳文件

file方法返回 illuminate/Http/UploadedFile類中的一個實例

獲取上傳的文件

$file = $request->file('photo'); file方法訪問

$file = $request->photo; 動態屬性來訪問

2種寫法。

hasFile 方法判斷請求中是否存在指定文件

$request->hasFile(‘xxx’) 如果文件存在(方法)

Validating Successful Uploads 驗證成功上傳

In addition to checking if the file is present, you may verify that there were no problems uploading the file via the isValid method:

if ($request->file('photo')->isValid()) {

//

}

文件路徑 & 擴展名

UploadedFile 類還包含訪問文件的全路徑和擴展名的方法。 extension 方法會根據文件內容判斷文件的擴展名。該擴展名可能會和客戶端提供的擴展名不同

$path = $request->photo->path();

\*\*\* 這個path是文php的 temp file path

$extension = $request->photo->extension();

存儲上傳文件

要存儲上傳的文件，先配置好 文件系統。

你可以使用 UploadedFile 的 store 方法把上傳文件移動到你的某個磁盤上，該文件可能是本地文件系統中的一個位置，甚至像 Amazon S3 這樣的雲存儲位置。

store 方法接受相對於文件系統配置的存儲文件根目錄的路徑。這個路徑不能包含文件名，因為系統會自動生成唯一的 ID 作為文件名。

store 方法還接受可選的第二個參數，用於存儲文件的磁盤名稱。這個方法會返回相對於磁盤根目錄的文件路徑：

$path = $request->photo->store('images');

$path = $request->photo->store('images', 's3');

\*\*\*

如果你不想自動生成文件名，那麼可以使用 storeAs 方法，它接受路徑、文件名和磁盤名作為其參數：

$path = $request->photo->storeAs('images', 'filename.jpg');

$path = $request->photo->storeAs('images', 'filename.jpg', 's3');

我們把上傳文件都放到public/uploads下

$file\_extension = $request->file->extension();

$file\_mime\_type = $request->file->getClientMimeType();

$original\_file\_name = $request->file->getClientOriginalName();

Other File Methods

There are a variety of other methods available on UploadedFile instances. Check out the API `(Taylor made a link here)` documentation for the class for more information regarding these methods.

storeAs 的根目錄是 /storage/app

比如:

$path = $request->avatar->storeAs('/public/uploads',$request->avatar->getClientOriginalName());

所以存到的路徑就會是 /storage/app/public/uploads



公共磁盤

public 磁盤適用於要公開訪問的文件。默認情況下， public 磁盤使用 local 驅動，並且將這些文件存儲在 storage/app/public 目錄下。為了使它們能通過網絡訪問，你需要創建從 public/storage 到 storage/app/public 的符號鏈接。

這種方式能把可公開訪問文件都保留在同一個目錄下，以便在使用零停機時間部署系統如 [Envoyer](https://envoyer.io/) 的時候，就可以輕松地在不同的部署之間共享這些文件。

你可以使用 Artisan 命令 storage:link 來創建符號鏈接：

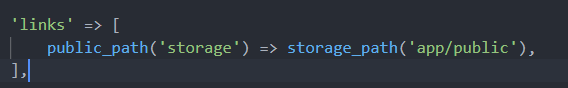
php artisan storage:link

一旦一個文件被存儲並且已經創建了符號鏈接，你就可以使用輔助函數 asset 來創建文件的 URL：

echo asset('storage/file.txt');

\*\*\*

您可以在您的 filesystems 配置文件中配置額外的符號鏈接。 每一個配置的鏈接創建時，將運行 storage:link 命令:



\*\*\*\*\*

如果是move方法, 當前路徑./ 就會是public

 $path = $request->avatar->move('./images',strstr($request->avatar->getClientOriginalName(),'.',true)  . md5(time() . rand(100000,999999)). '.' .$request->avatar->extension());

如果是storeAs方法, 當前路徑./就會是storage/app

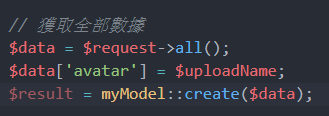
$path = $request->avatar->storeAs('./public/images',strstr($request->avatar->getClientOriginalName(),'.',true)  . md5(time() . rand(100000,999999)). '.' .$request->avatar->extension());

在Models/myModel.php 給$fillable加上avatar



\*\*\*\*\*

$request->all() 接收的數據可以直接插入數據庫



getErrorMessage()是file下的方法

if($request->avatar->getErrorMessage()){

                    echo $request->avatar->getErrorMessage();

                }

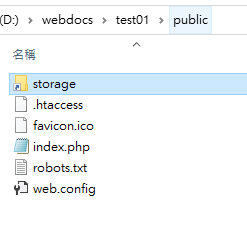
$request->avater 相當於 $request->file(‘avatar’);

getErrorMessage好似沒用了。

讀取public文件

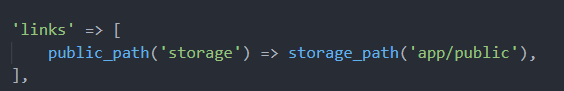
在項目根目錄下:

php artisan storage:link



會生成一個storage捷徑, 幫我們跑到storage/app/public下

可看config/filesystem.php文件



由於我們的images是放到storage/app/public/images下

所以訪問時的路徑是: storage/images/xxxxx.jpg

url路徑的storage幫我們跑到storage/app/public下

$target = myModel::select('\*')->orderBy('id','desc')->get();

$images = asset('storage/images/' . $target[0]->avatar);

數據分頁

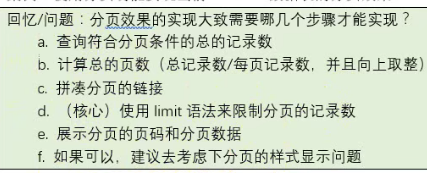
無分頁效果的列表展示

laravel直接調用模型的分頁方法, 返回對應的數據和分頁的字符串。

分頁類的位置

illuminate/Pagination的paginator.php這個方法

案例: 使用分頁功能實現當前member數據表的分頁效果.



\*\*\*

myModel::get() 這時get()方法就直接獲取數據庫內所有數據。

<img style="width: 100px" src="{{asset('/storage/images/' . $data->avatar)}}" alt="">

在config/filesystem.php文件內

/storage 就是根目錄(public) 下的storage,

而storage我們links到了storage/app/public

之後再找images/…..jpg

class Illuminate\Database\Eloquent\Builder

### ****paginate****(int|null $perPage = null, array $columns = ['\*'], string $pageName = 'page', int|null $page = null)



$datas = myModel::paginate(5);

每頁5條數據

之後需要在頁面中出現 分頁連接

用paginator的links()方法

public function links($view = null, $data = [])

    {

        return $this->render($view, $data);

    }

而且要加上boot方法改善樣式(laravel8的 bootstrap問題)

For laravel 8 and above, simply search for AppServiceProvider.php and add these codes into it:

use Illuminate\Pagination\Paginator;

public function boot()

{

Paginator::useBootstrap();

}

找到app/provider/appServiceProvider.php

加上

use Illuminate\Pagination\Paginator;

 public function boot()

    {

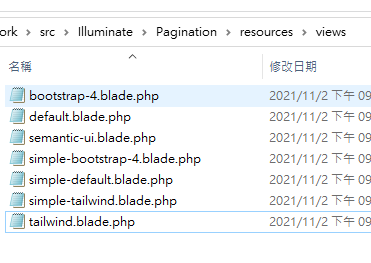
*//*

        Paginator::useBootstrap();

    }

paginate的樣式是寫死的, 如果要改, 要到Illuminate\Pagination\resources\views

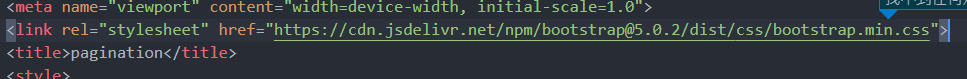
以下是分頁paginate->links() 的樣式



使用方法如下:

->links(‘pagination::tailwind’);

->links(‘pagination::bootstrap-4’); 等…



搞了1小時, 因為無法用到bootstap樣式, 我們直接在頁面上加上bootstrap5的樣式連結。

成功了….



Paginator::useBootstrap();

是一定要加的。

驗證碼

原生php



到packagist.org

search: captcha

completely automated public turning test to tell computers and human aprat

\*\*\*

packagist.org 是一個下載php包的網站

我們這裡用

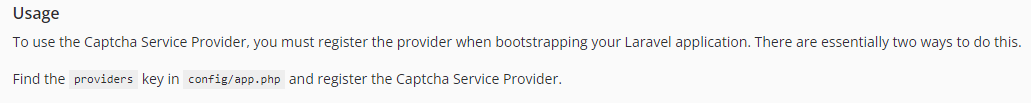
mews/captcha

Laravel 5 & 6 Captcha Package 因為大陸不能用google的recaptcha(最多人用)

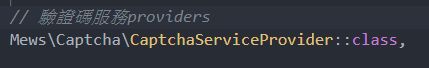
安裝:

composer require mews/captcha

laravel第三方包放在vendor中



在providers下 加上

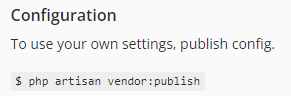




在aliases下加上



如果需要定義自己的配置, 可生成配置文件



php artisan vendor:publish



我們是想生成11號 captcha的config

之後在config/就會有captcha.php

案例: 我們在之前的verify\_form中加入驗證碼

Mews/captcha用法在<https://packagist.org/packages/mews/captcha>

用了Mews/captcha 我們會多了一個validator的rules 名為captcha



echo '<pre>';

            print\_r(request()->all());

我們查看request內的數據

看到captcha數據 傳為了:captcha (我們在html name屬性寫錯了)

request()->all() 以數組形式返回所有$\_POST/$\_GET數據

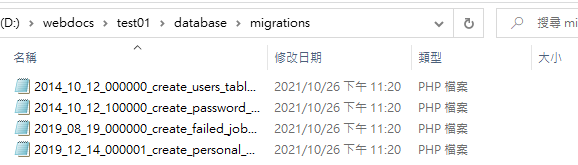
數據表的遷移與填充 (多用戶teamwork時會用到)

遷移:創建數據表的操作+刪除數據表的操作

填充:往數據表里填充寫入測試數據(數據插入)

遷移文件默認放到database\migrations下

系統自帶4個遷移文件



案例: 需要創建試卷的數據表, 假設數據表的名字叫做paper, 遷移文件名:create\_paper\_table

遷移就像是對數據庫進行的版本控制，讓你的團隊能夠輕松地去定義和共享程序的數據庫結構。遷移通常配合 Laravel 的結構生成器，可以輕松生成應用程序的數據庫結構。如果團隊中有個成員在他本地的數據庫環境中手動的添加了某個字段，那麼你將會面對如何解決數據庫遷移的問題。

Laravel 的 Schema facade 提供了數據庫相關的支持，可以在所有 Laravel 支持的數據庫管理系統中創建和操作表。

生成遷移

使用　make:migration Artisan 命令 命令來創建遷移：

php artisan make:migration create\_users\_table

php artisan make:migration 遷移文件名

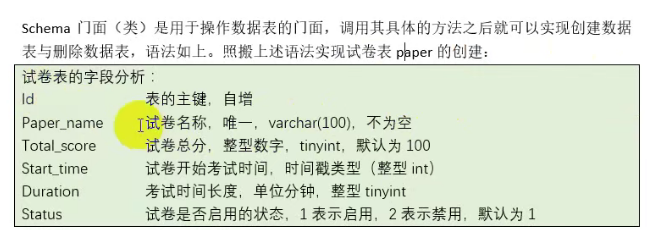
php artisan make:migration create\_paper\_table

migration遷移文件下的up方法與down方法

up 的作用是向數據庫中添加新的表或列或索引，

而 down 的方法，就是 up 方法的反向操作。

在這兩種方法中，您可以使用 Laravel 的 Schema 生成器顯式地創建和修改表。要瞭解 Schema 生成器上可用的所有方法，請查看其文檔。



Status 我們可以用 enum或tinyint

$table表示整個表的實例

語法:

$table->列類型方法(字段名,[長度/值範圍]) -> 列修飾方法([修飾的值]);

可用的字段類型

數據庫結構生成器包含構建表時可以指定的各種字段類型：

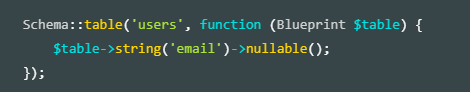


查看<https://learnku.com/docs/laravel/8.x/migrations/9403#creating-tables>

列修飾方法



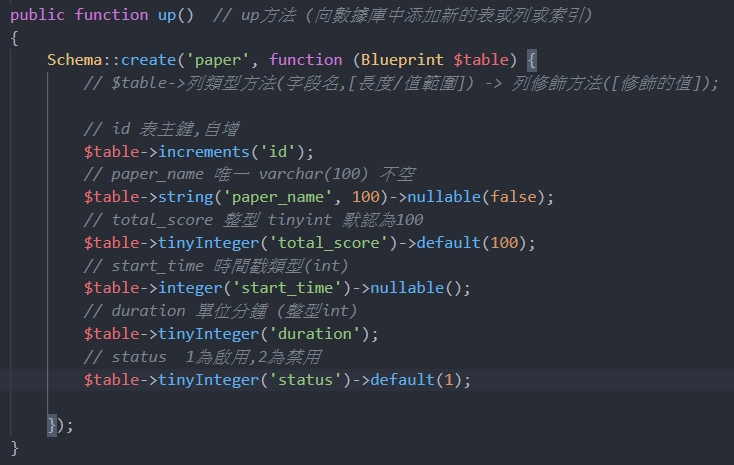
例字:



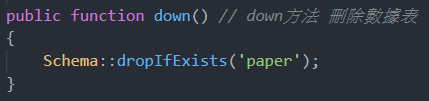
$table->string(‘email’)->nullable();

創建數據表的代碼 up()

Schema::create(‘paper’,function(Blueprint $table){…})



刪除數據表的代碼 down() Schema::dropIfExists(‘paper’);



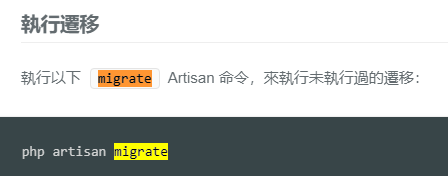
結構生成器支持多種類型的索引。下面的例子中新建了一個值唯一的 email 字段。我們可以將 unique 方法鏈式地添加到字段定義上來創建索引：

$table->string('email')->unique();

\*\*\*

執行遷移文件

執行分別為up和down的方法



php artisan migrate

有些遷移操作是破壞性的，這意味著它們可能會導致數據丟失。為了防止您對生產數據庫運行這些命令，在執行這些命令之前，系統將提示您進行確認。要在運行強制命令的時候去掉提示，請使用 -force 標志：

php artisan migrate --force

回滾遷移 要回滾到最後一次操作，你可以使用 rollback 命令。此命令會回滾到最後 「一批」 的遷移，這可能會包含多個遷移文件：

php artisan migrate:rollback

通過向 rollback 命令加上 step 參數，可以回滾指定數量的遷移。例如，以下命令將回滾最後五個遷移：

php artisan migrate:rollback --step=5

如果您在遷移後覺得有誤，可以使用重置命令 migrate:reset，回滾剛才進行的所有遷移操作，以恢復到上一次遷移狀態。

php artisan migrate:reset

創建字段

使用 Schema facade 的 table 方法可以更新現有的數據表。如同 create 方法一樣，table 方法會接受兩個參數：一個是表名，另一個則是接收可以用來向表中添加字段的 Blueprint 實例的閉包：

Schema::table('users', function (Blueprint $table) {

$table->string('email');

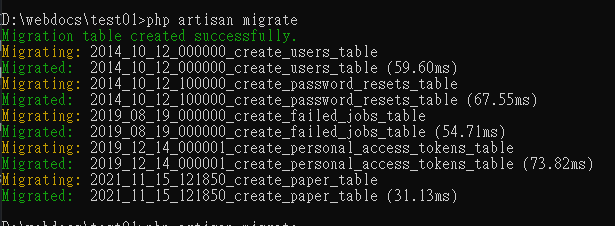
});

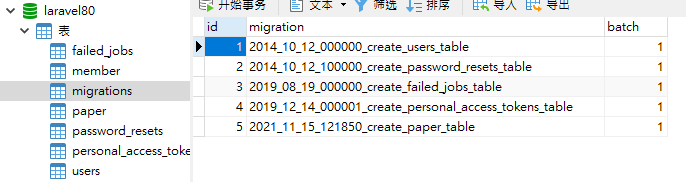
\*\*\*

我們執行

php artisan migrate

會執行未遷移過的遷移文件





有4個是laravel系統自帶的數據表

php artisan migrate:rollback　就是刪除數據表 (不會刪除遷移文件, 只刪除遷移執行記錄)

php artisan migrate

\*\*\*

rollback精確來說不是刪除數據表 而是回滾到上一次的操作(未生成數據表)。

理論上rollback是執行down()方法 所以就是dropIfExists

php artisan migrate會執行 未執行的遷移文件的up()方法。



數據表填充器

填充器文件的創建 (填充器 = 種子文件)

Laravel 包含一個填充類可以為你的數據庫填充測試數據，所有的填充類都放在 database/seeds 目錄下。你可以隨意為填充類命名，但是更建議您遵守類似 UserSeeder 的命名規范。通常， Laravel 默認定義了一個 DatabaseSeeder 類。通過這個類，你可以用 call 方法來運行其它的 seed 類從而控制數據填充的順序。

運行 [Artisan 命令](https://learnku.com/docs/laravel/8.x/artisan) make:seeder 生成 Seeder，框架生成的 seeders 都放在 database/seeds 目錄下：

php artisan make:seeder 填充器名

命名規則: 大寫表名+TableSeeder

以paper為例: 就是

php artisan make:seeder PaperTableSeeder

\*\*\*

seeder 類只包含一個默認方法：run 。這個方法會在執行 db:seed 這個 Artisan 命令 時被調用。在 run 方法裡你可以根據需要在數據庫中插入數據。你也可以用 查詢構造器 或 Eloquent 模型工廠 來手動插入數據。

技巧：使用數據填充時會自動禁用 [批量賦值保護](https://learnku.com/docs/laravel/8.x/eloquent#mass-assignment)。

run() 中可以使用DB門面, 但是注意, DB門面在使用時不需要用戶自己引入, 一旦引入則報錯,可以直接使用。

DB::table

table方法, 插入數據到哪張表

運行 Seeders

你可以使用 Artisan 命令 db:seed 來填充數據庫了。默認情況下，db:seed 命令將運行 DatabaseSeeder 類，這個類可以用來調用其它 Seed 類。不過，你也可以使用 --class 選項來指定一個特定的 seeder 類：



php artisan db:seed

php artisan db:seed --class=PaperTableSeeder

由於我們的paper\_name是unique, 所以無法重複插入數據。

Eloquent ORM 總結操作 \*\*\*\*\*

1.先自己創建一張數據表

2.生成myModel模型里面定義好

創建模型最簡單的方法就是使用 make:model [Artisan 命令](https://learnku.com/docs/laravel/8.x/artisan)：

php artisan make:model home/myModel

protected $table = 'member';

    protected $primaryKey = 'id';

    public $timestamps = false;

    protected $fillable = ['id','name','age','email','avatar'];

3.生成myModelController

在myModelController控制器中 引入myModel模型

use App\Models\home\myModel;

4.之後就可以在控制器中寫curd方法

public function myModel\_add(Request $request){

*// 實例化myModel模型 (插入新數據一定要用實例化)*

        $member = new myModel;

*// 這裡我們不想寫表單去接收數據, 就直接給$request接收的數據賦值*

        $request->name = 'peter';

        $request->age = 30;

        $request->email = 'myModel\_add\_testing@gamil.com';

*// 接收請求的對應數據 放到$member對應字段*

*// $member->id = $request->id; //id不需要, 因為是auto\_increment*

        $member->name = $request->name;

        $member->age = $request->age;

        $member->email = $request->email;

        $member->save();

    }

項目初始化 (使用laravel項目) TestProject

1. 創建laravel項目

選定要創建項目的位置

輸入composer命令

composer create-project --prefer-dist laravel/laravel ./${project.name}

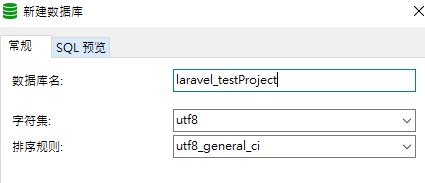
如果沒有${project.name} 就直接在目錄下生成所有項目文件。

我們在D:/webdocs下

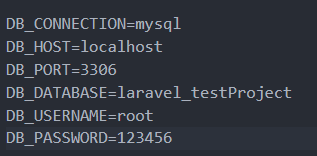
composer create-project --prefer-dist laravel/laravel ./testProject

1. 建立數據庫

創建數據庫給項目用(navicat) laravel\_testProject



修改.env文件,(或config.database文件)



1. 設置網站本地化為中文 (語言包) (可選操作)

語言包放到resources/lang下

到config/app 的local 改為對應的language

4 設置項目時區

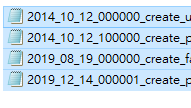
config/app下的timezone 改為

'timezone' => 'Asia/Macau',

5.清理項目 (刪除不需要的文件)

刪除app/Http/Controllers/Auth目錄(5.x version)

刪除database/migration 不需要的遷移文件



刪除database/seeders內的DatabaseSeeder.php

刪除resources/views/welcome.blade.php

public/js與public/css也刪除(5.x version)

6. 關閉mysql 嚴格模式

config/database.php內



strict 改為false

嚴格模式說明功能:

不支持對not null字段插入null值

不支持對自增長字段插入”值

不支持text字段有默認值

7.安裝debugbar工具條(可選)

條件php>=7.0

packagist.org search: debugbar

我們用barryvdh/laravel-debugbar

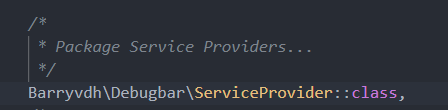
composer require barryvdh/laravel-debugbar --dev

修改config/app.php文件進行配置 provider與aliases

### Laravel without auto-discovery:

If you don't use auto-discovery, add the ServiceProvider to the providers array in config/app.php

加到providers內的 package service providers



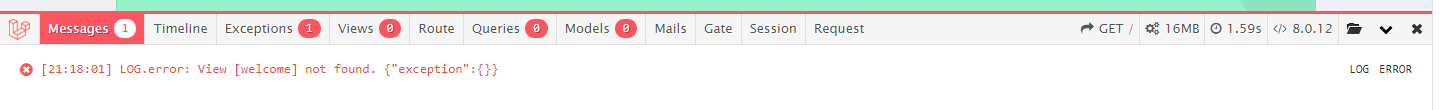
Barryvdh\Debugbar\ServiceProvider::class,

If you want to use the facade to log messages, add this to your facades in app.php:

加到Aliases

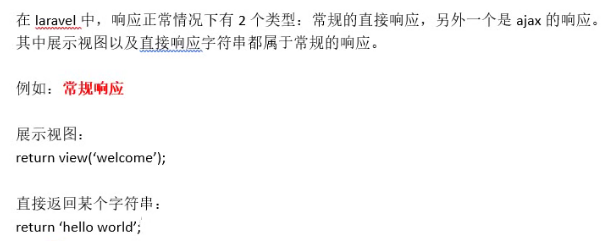
'Debugbar' => Barryvdh\Debugbar\Facades\Debugbar::class, ()

之後用php artisan server



這就是debugbar

6 響應處理



ajax請求響應

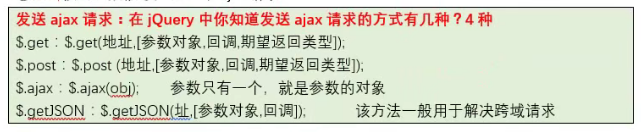
常見的ajax響應數據類型: json , xml , text/html。

JSON用最多。

json\_encode函數



案例中我們用jQuery實現ajax請求



$.get(url,[data],[callback],[type]) \*\*\*data是返回的數據 數組形式\*\*\*

$.post(url,[data],[callback],[type])

$.ajax(obj)

$.getJSON(url,[data],[callback])

可選參數可跳過 不用占位(每個可選參數都不用類型)

\*\*\*重點

laravel框架中的響應json數據方法:

return response()->json($data);

2種響應數據方式

1. return json\_encode($data);

2. return response()->json($data); laravel方法

\*\*\*

補充: laravel中bool值是不能被直接通過return響應輸出的。

return true;

如果 響應數組, 會自動轉換為 Object數據 (php數組比較特別的(鍵值對)只能轉為json對象的)

return ['a'=>123];

跳轉響應(重定向)

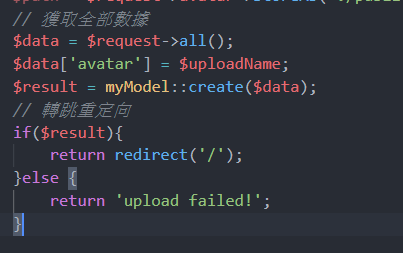
3種跳轉方式

return redirect(路由地址);

return redirect()->to(路由地址);

return redirect()->route(路由別名); 需要路由有別名

在之前的upload\_avatar案例中加入 return redirect(‘/’);



會話控制(了解)

常見應用 – curd session

session默認存到文件中

storage/framework/sessions

session配置

Session 的配置文件存儲在 config/session.php 文件中。請務必查看此文件中對於你而言可用的選項。默認情況下，Laravel 為絕大多數應用程序配置的 Session 驅動為 file 。在生產環境中，你可以考慮使用 memcached 或 redis 驅動，讓 Session 的性能更加出色。

Session driver 的配置預設了每個請求存儲 Session 數據的位置。Laravel 自帶了幾個不錯而且開箱即用的驅動：

file - 將 Session 存儲在 storage/framework/sessions 中。

cookie - Sessions 被存儲在安全加密的 cookie 中。

database - Sessions 被存儲在關系型數據庫中。

memcached / redis - Sessions 被存儲在基於高速緩存的存儲系統中。

array - Sessions 存儲在 PHP 數組中，但不會被持久化。

技巧：數組驅動一般用於 測試 並且防止存儲在 Session 中的數據被持久化。

使用 Session

獲取數據

Laravel 中處理 Session 數據有兩種主要方法：全局輔助函數 session 和通過一個 Request 實例。首先，我們來看看通過控制器方法類型提示一個 Request 實例來訪問 session。控制器方法依賴項會通過 Laravel 服務容器 實現自動注入：

public function show(Request $request, $id)

{

$value = $request->session()->get('key');

//

}

當你從 Session 獲取值時，你還可以傳遞一個默認值作為 get 方法的第二個參數。如果 Session 中不存在指定的鍵，便會返回這個默認值。若傳遞一個閉包作為 get 方法的默認值，並且所請求的鍵並不存在時，get 方法將執行閉包並返回其結果：

$value = $request->session()->get('key', 'default');

全局輔助函數 Session

你也可以使用全局的 PHP 輔助函數 session 來獲取和存儲 Session 數據。 使用單個字符串類型的值作為參數調用輔助函數 session 時，它會返回該字該符串對應的 Session 鍵的值。當使用一個鍵值對數組作為參數調用輔助函數 session 時，傳入的鍵值將會存儲在 Session 中：

// 獲取 session 中的一條數據...

$value = session('key');

// 指定默認值

$value = session('key', 'default');

存儲數據

想要存儲數據到 Session，你可以使用 put 方法，或者使用輔助函數 session：

// 通過請求實例...

$request->session()->put('key', 'value');

// 通過全局輔助函數...

session(['key' => 'value']);

保存數據到 Session 數組中

push 方法可以將一個新的值添加到 Session 數組內。例如，假設 user.teams 這個鍵是包括團隊名稱的數組，你可以這樣將一個新的值加入到數組中：

$request->session()->push('user.teams', 'developers');

檢索 & 刪除一條數據

pull 方法可以只使用一條語句就從 Session 中檢索並刪除一條語句：

$value = $request->session()->pull('key', 'default');

閃存數據

有時候你可能想在 Session 中保存數據用於下一次請求，這時你可以使用 flash 方法。使用這個方法保存在 Session 中的數據，只會保留到下一個 HTTP 請求到來之前，然後就會被刪除。閃存數據主要用於短期的狀態消息：

$request->session()->flash('status', 'Task was successful!');

如果你需要在更多的請求中使用到該一次性數據，你可以使用 reflash 方法，該方法會將所有一次性請求保留到下一次請求。如果你想保存一次性數據，你可以用 keep 方法：

$request->session()->reflash();

$request->session()->keep(['username', 'email']);

刪除數據

forget 方法會從 Session 中刪除指定數據，如果想從 Session 中刪除所有數據，可以使用 flush 方法：

// 刪除單個值...

$request->session()->forget('key');

// 刪除多個值...

$request->session()->forget(['key1', 'key2']);

$request->session()->flush();

重新生成 Session ID

重新生成 session ID 通常是為了防止惡意用戶利用 session fixation 對你的應用進行攻擊。 如果你使用了內置的 LoginController，Laravel 會自動重新生成身份認證中的 Session ID。否則，你需要手動使用 regenerate 方法重新生成 Session ID。

$request->session()->regenerate();

獲取所有 Session 數據 如果你想獲取 session 中的所有數據，可以使用 all 方法：

$data = $request->session()->all();

判斷 Session 中是否存在某個值

要確定 Session 中是否存在某個值，可以使用 has 方法。如果該值存在且不為 null，那麼 has 方法會返回 true：

if ($request->session()->has('users')) {

//

}

要確定 Session 中是否存在某個值，即使其值為 null，也可以使用 exists 方法。如果值存在，則 exists 方法返回 true：

if ($request->session()->exists('users')) {

//

}

*// 通過laravel方法的session無法用原生php 的$\_SESSION 獲取*

*// session\_start();*

*// dd($\_SESSION['name']);*

laravel $request->session()的

增 put (全新的鍵值push, push session的值是數組類型)

刪 forgot (全刪flush)

查 get all

改 put

緩存操作

Laravel 為各種後端緩存提供了豐富而統一的 API，其緩存配置位於 config/cache.php 文件中。在該文件中你可以指定應用默認使用哪個緩存驅動。Laravel 支持當前流行的後端緩存，例如 Memcached 和 Redis。

緩存配置文件還包含各種其他選項，這些選項都記錄在文件中，因此請確保閱讀這些選項。默認情況下，Laravel 配置為使用 file 緩存驅動，它將序列化的緩存對象存儲在文件系統中。對於較大型應用，建議使用更強大的驅動程序，例如 Memcached 或 Redis。你甚至可以為同一個驅動程序配置多個緩存配置。

config/cache



默認的cache driver是file

如果想改cache driver 可以到.env改

我們是用file來驅動緩存

當使用 database 緩存驅動時，你需要配置一個表來存放緩存數據。下面是構建緩存數據表結構的 Schema 聲明示例：

Schema::create('cache', function ($table) {

$table->string('key')->unique();

$table->text('value');

$table->integer('expiration');

});

#### Redis

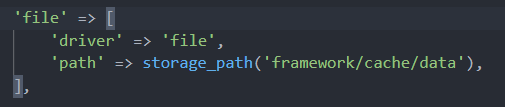
在使用 Laravel 的 Redis 緩存之前，你需要通過 PECL 安裝 PhpRedis PHP 擴展，或者通過 Composer 安裝 predis/predis 包（~1.0）。

緩存使用

獲取緩存實例

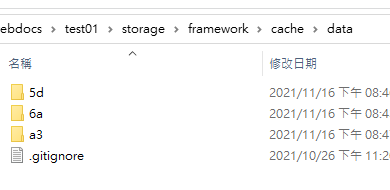
Illuminate\Contracts\Cache\Factory 和 Illuminate\Contracts\Cache\Repository 契約 提供了 Laravel 緩存服務的訪問機制。 Factory 契約為你的應用程序定義了訪問所有緩存驅動的機制。 Repository 契約通常是由你的 cache 配置文件指定的默認緩存驅動實現的。

當然，你也可以使用 Cache Facade，我們將在後續的文檔中介紹。Cache Facade 為 Laravel 緩存契約底層的實現提供了方便又簡潔的方法：



cache文件是放到storage/framework/cache/data下的

每一個緩存都會在data中生成一個file



use Illuminate\Support\Facades\Cache;

class UserController extends Controller

{

/\*\*

\* 顯示該應用程序的所有用戶的列表

\*

\* @return Response

\*/

public function index()

{

$value = Cache::get('key');

//

}

}

訪問多個緩存存儲

使用 Cache Facade，你可以通過 store 方法來訪問各種緩存存儲。傳入 store 方法的鍵應該對應 cache 配置信息文件中的 stores 配置數組中所列出的一個：

$value = Cache::store('file')->get('foo');

Cache::store('redis')->put('bar', 'baz', 600); // 10 Minutes

從緩存中獲取數據

Cache Facade 的 get 方法用於從緩存中獲取數據。如果該數據在緩存中不存在，那麼該方法將返回 null 。正如你想的那樣，你也可以向 get 方法傳遞第二個參數，用來指定如果查找的數據不存在時你希望返回的默認值：

$value = Cache::get('key');

$value = Cache::get('key', 'default');

你甚至可以傳遞一個 Closure 作為默認值。如果指定的數據在緩存中不存在，將返回 Closure 的結果。傳遞閉包的方法允許你從數據庫或其他外部服務中獲取默認值。

$value = Cache::get('key', function () {

return DB::table(...)->get();

});

檢查緩存項是否存在

has 方法可以用於判斷緩存項是否存在。如果值為 null，則該方法將會返回 false：

if (Cache::has('key')) {

//

}

在緩存中存儲數據

你可以使用 Cache Facade 的 put 方法將數據存儲到緩存中：

Cache::put('key', 'value', $seconds);

$seconds是緩存時間的秒數

如果緩存的過期時間沒有傳遞給 put 方法， 則緩存將永久有效：

Cache::put('key', 'value');

除了以整數形式傳遞過期時間的秒數，你還可以傳遞一個 DateTime 實例來表示該數據的到期時間：\*\*\*

Cache::put('key', 'value', now()->addMinutes(10));

只存儲沒有的數據

add 方法將只存儲緩存中不存在的數據。如果存儲成功，將返回 true ，否則返回 false ：

Cache::add('key', 'value', $seconds);

數據永久存儲

forever 方法可用於持久化將數據存儲到緩存中。因為這些數據不會過期，所以必須通過 forget 方法從緩存中手動刪除它們：

Cache::forever('key', 'value');

forever 設置unix時間為99999999 就是2286年才結束數據的生命周期



從緩存中刪除數據

你可以使用 forget 方法從緩存中刪除這些數據：

Cache::forget('key');

你也可以通過提供零或者負的 TTL 值來刪除這些數據：

Cache::put('key', 'value', 0);

Cache::put('key', 'value', -5);

你可以使用 flush 方法清空所有的緩存：

Cache::flush();

注意：清空緩存的方法並不會考慮緩存前綴，會將緩存中的所有內容刪除。因此在清除與其它應用程序共享的緩存時，請慎重考慮。

Cache 輔助函數

除了可以使用 Cache 的門面以及 Cache 契約 外，你也可以使用全局輔助函數 cache 來獲取和保存緩存數據。當只使用一個字符串參數調用 cache 函數時，這將返回給定鍵對應的值。

$value = cache('key');

如果你向函數提供了一組帶有過期時間的鍵值對，那麼在這段時間內，它將緩存此數據。

cache(['key' => 'value'], $seconds);

cache(['key' => 'value'], now()->addMinutes(10));

遞增與遞減值

increment 和 decrement 方法可以用來調整緩存中整數項的值。這兩個方法都可以傳入第二個可選參數，這個參數用來指明要遞增或遞減的數量：

Cache::increment('key');

Cache::increment('key', $amount);

Cache::decrement('key');

Cache::decrement('key', $amount);

\*\*\*\*

我們一用increment, decrement (問題多多)

cache生命周期馬上更新到2014年的時間, cache直接死亡。

我們的increment, decrement只跑第一次, 第2次取api接口就不動了。

\*\*\*\*

重點:

如果使用increment(decrement)時, 對應的cache數據\*\*\*一定要有生命周期\*\*\*

也不能是forever生成的cache數據。 \*\*\*如果沒有生命周期, 就不能用increment/decrement

不然increment, decrement只跑第一次, 第2次取api接口就不動了。

獲取和存儲

有時你可能想從緩存中獲取一個數據，而當請求的緩存項不存在時，程序能為你存儲一個默認值。例如，你可能想從緩存中獲取所有用戶，當緩存中不存在這些用戶時，程序將從數據庫將這些用戶取出並放入緩存。你可以使用 Cache::remember 方法來實現：

$value = Cache::remember('users', $seconds, function () {

return DB::table('users')->get();

});

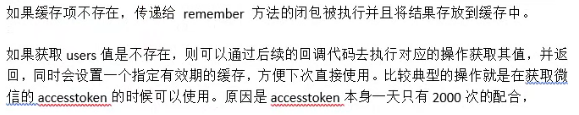
如果緩存中不存在你想要的數據時，則傳遞給 remember 方法的 閉包 將被執行，然後將其結果返回並放置到緩存中。

你可以使用 rememberForever 方法從緩存中獲取數據或者永久存儲它：

$value = Cache::rememberForever('users', function () {

return DB::table('users')->get();

});



Cache::remember 可用於生成限時token

cache中的方法

put 插入數據/更新

add 只插入,如果而存在,不會更新

forever 生成永久時間周期cache數據

get 獲取數據

forget 刪除數據

remember 獲取與存儲數據, 如果存在, 不會更新

如生命周期結束, 數據就不存在了。

聯表查詢

聯表要求至少有2張表, 并且還是存在關系的2張表。

例如:

文章表(article)

id 主鍵

article\_name 文章名

author\_name 作者id

作者表 (author)

id 主鍵

author\_name 作者名



我們用創建遷移文件

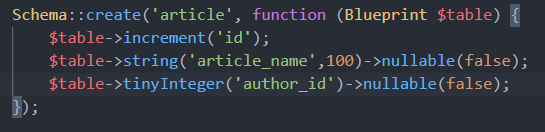
php artisan make:migration create\_article\_table

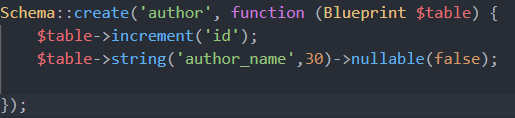
php artisan make:migration create\_author\_table

在database/migration下



編寫遷移文件





執行migration文件

php artisan migrate

模擬數據(利用填充器實現, 可以將多個數據表的寫入操作寫在一起)

php artisan make:seeder ArticleAndAuthorTableSeeder



執行seeder文件

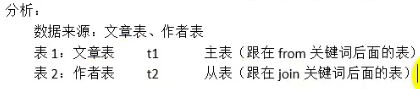
php artisan db:seed --class=ArticleAndAuthorTableSeeder

現在有數據了



查詢出的(article表中的author\_id)會獲取對應的(author表中的author\_name)

分析



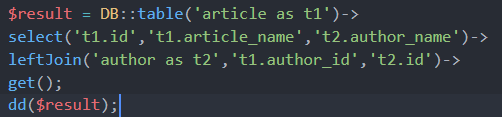


SQL語句 (外連接 左合并)

select t1.id,t1.article\_name,t2.author\_name from article as t1 left join author as t2 on t1.author\_id = t2.id;

以上SQL語句轉為DB門面的鏈式代碼

創建一個路由與view



關聯模型(重點+難點)

關聯模型就是綁定模型(表)的關系, 後續需要使用聯表的時候就可以直接使用關聯模型。(關聯模型需要創建模型)

一對一關系

php artisan make:model home/article

php artisan make:model home/author

關聯模型的方法(重點)

先分析 主表article 和 從表author

article關聯author

關聯代碼寫在主模型article中。

寫一個方法, 方法名是被關聯的模型的模型名

我們要在 article 模型中寫一個 author 方法。 在 author 方法內部調用 hasOne 方法並返回其結果：

public function author()

    {

        return $this->hasOne('App\Models\home\author','id','author\_id');

    }

一旦定義了模型關聯，我們就可以使用 Eloquent 的動態屬性獲得相關的記錄。動態屬性允許你訪問關聯方法就像訪問模型中定義的屬性一樣：

$phone = User::find(1)->phone;

hasOne(被關聯模型的命名空間 , [外鍵(被關聯模型的表的外鍵)] , [本地鍵(本模型的關聯鍵)])

定好關聯模型, 我們就可以寫api接口 編歷數據

記得引入模引 use….

use App\Models\home\article;

use App\Models\home\author;

原生獲取另一張表的數據

author::find($value->author\_id)->author\_name

定好關聯模型,就可以動態獲取 (而且不需要use被關聯的引入模型)

$value->author->author\_name

關聯模型就是使我們多了一個動態調用方法

$value->author->author\_name

author 就是關聯方法

定義反向關聯

我們就可以使用與 hasOne 方法對應的 belongsTo 方法來定義反向關聯：

public function user() {

return $this->belongsTo('App\Models\User', 'foreign\_key', 'other\_key');

}

一對多關系 (一篇文章有多個評論)

comment table:

id , comment, article\_id

php artisan make:migration create\_comment\_table

寫migration文件

Schema::create('comment', function (Blueprint $table) {

            $table->increments('id');

            $table->string('comment')->nullable(false);

            $table->tinyInteger('article\_id');

        });

執行migration 生成表

php artisan migrate

創建填充器

php artisan make:seeder CommentTableSeeder

寫seeder文件



執行seeder

php artisan db:seed --class=CommentTableSeeder

創建comment模型

php artisan make:model home/comment

定義模型

protected $table = 'comment';

public $timestamps = false;



在article model中

*// 一對多, 加入comment關聯*

hasMany(‘被關聯模型’,’foreign key’ , ‘local\_key’);

public function comment(){

        return $this->hasMany('App\Models\home\comment','article\_id','id');

    }

之後寫api接口 onetomany

public function OneToMany(){

        $datas = article::get();

*// 循環編歷*

        echo 'id' . '&emsp;' . 'article\_name' . '&emsp;' . 'author\_name' . '&emsp;' . 'comment' .'<br>';

        foreach($datas as $key => $value){

            echo $value->id . '&emsp;&emsp;' . $value->article\_name . '&emsp;&emsp;&emsp;' .

            $value->author->author\_name . '這文章的評論為:' . '<br>';

*// 原生寫法(無關聯模型下)*

*// $comments = comment::where('article\_id' , $value->id)->get();*

*// foreach($comments as $v){*

*//     echo $v->comment . '<br>';*

*// }*

            foreach($value->comment as $k => $v){  *// comment.article\_id 與 article.id關聯了*

*// $value->comment會取得對應$value的article.id內的comment.comment(關聯了)*

*// $value->comment 是數組 (有些文章一篇article有多個comment)*

                echo $v->comment . '<br>';

            }

        }

    }

多對多關系

例如:

一篇文章有多個關鍵詞, 一個關鍵詞可能被多個文章使用

需要依靠第三張表建立關系(關系表)。

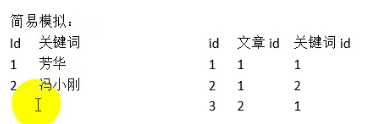
創建 關鍵詞表與 關系表

關鍵詞表

id , keyword

關系表

id , article\_id , key\_id



php artisan make:migration create\_keyword\_table

php artisan make:migration create\_relation\_table

Schema::create('keyword', function (Blueprint $table) {

            $table->increments('id');

            $table->string('keyword',10)->nullable(false);

        });

Schema::create('relation', function (Blueprint $table) {

            $table->increments('id');

            $table->tinyInteger('article\_id');

            $table->tinyInteger('key\_id');

        });

執行migration文件

php artisan migrate

生成填充器

php artisan make:seeder KeywordAndRelationTableSeeder

執行填充器

php artisan db:seed --class=KeywordAndRelationTableSeeder

創建模型

關聯多對多

先創建keyword的模型

php artisan make:model home/keyword

protected $table = 'comment';

public $timestamps = false;

模型結構

多對多關聯通過調用 belongsToMany 這個內部方法返回的結果來定義

除了自定義連接表的表名，你還可以通過傳遞額外的參數到 belongsToMany 方法來定義該表中字段的鍵名。第三個參數是定義此關聯的模型在連接表裡的外鍵名，第四個參數是另一個模型在連接表裡的外鍵名：

4個參數

return $this->belongsToMany(

‘被關聯模型的路徑’ ,

多對多模型關系表表名,

此關聯的模型在連接表裡(關系表)的外鍵名 ,

另一個模型(被關聯模型)在連接表(關系表)裡的外鍵名);

public function keyword(){

        return $this->belongsToMany('App\Models\home\keyword','relation','article\_id'.'key\_id');

    }

多對多關聯 會自動把對應的外鍵與id 關聯\*\*\*\*\*

編寫api接口 編歷輸出結果

 public function ManyToMany(){

        $datas = article::get();

        foreach($datas as $key => $value){

            echo '當前文章名為: ' . $value->article\_name . '其所用的關鍵詞為:' . '<br/>' ;

            foreach($value->keyword as $k => $v){

                echo $v->keyword . ' <br>';

            }

        };

    }