Laravel 5.5 入门教程 之 艺术家的框架

By IT崖柏图
Mail 973714522@qq.com
出自 布尔教育PHP高端教育培训

第1章 如何安装laravel

1.1 安装方法及理解

我们该如何安装laravel,如果从网上搜索的话,你大概能够找到以下方法,但是不要去死记.
而是要从原理去推出:
`完成的 laravel = laravel本身 + composer指定的依赖库`

所以你至少可以有这四种方法:
1 . 用`composer create-project` 命令自动下载 `laravel`, 同时自动安装依赖库
** `composer create-project laravel/laravel=5.5`**

2 . 手动下载 laravel 本身 ,composer 安装依赖库 (半自动化)
https://github.com/laravel/laravel/tree/v5.5.0
下载laravel,再到项目目录下,执行`composer install`;

知道怎么安装？
答：下载后解压。看看解压的目录里是否有.env文件，没有的话则拷贝.env.example文件并重命名为.env。然后cmd进入项目目录，然后执行php artisan key:generate命名。

3 . 下载别人帮拼装好的laravel本身+composer中指定的库，不需要安装composer；
此方法的局限性在于，如果项目过程中需要其他的库，还是需要`composer`安装；

4 . laravel 安装器,可以帮你完成这两步(强烈不推荐，麻烦而且不认识国内镜像源)

```
# 安装 “laravel 安装器 ” (不是 laraval )
composer global require "laravel/installer"
 - cd /usr/local/nginx/html
~/.composer/vendor/bin/laravel new <you appName>
```

5 . 集成包环境安装
Homestead虚拟机安装 https://laravelacademy.org/post/7658.html
Mac下Valet安装laravel https://laravelacademy.org/post/7678.html

1.2 配置虚拟主机

注意,在项目路径`public`下

修改虚拟主机配置文件，在`apache`添加如下代码：

```
<VirtualHost *:80>
DocumentRoot "D:/xampp/htdocs/<project>/public"
ServerName ddd.com
</VirtualHost>
```

`host文件 127.0.0.1 ddd.com`

第2章 路由器

路由简介

1,简单的说就是将用户的请求转发给相应的程序去处理
2,作用建立url和程序之间的映射
3,请求类型get,put,post,patch,delete等
任何框架都离不开路由器，TP是通过地址栏规则生成，如：`xxx.com/home/user/add`;

2.1 路由器如何调用控制器

laravel的路由器与控制器的关系,需要明确的在`<project>/routes/web.php`
文件中明确定义.
格式如下：

```
基础路由
/*
当用 GET 方式访问 xx.com/yy 这个地址的时候用匿名函数去响应 .
*/
Route::get('/yy', function(){
 return '123';
});

/*
当用 POST 方式访问 xx.com/zz 这个地址时,用 匿名函数去响应 .
*/
Route::post('/zz', function(){
 return '123';
});

/*
当 GET 访问网站根目录 "/" 时,用第2个参数的匿名函数去响应 .
*/
Route::get('/', function () {
    return 'hello';
})

多请求路由
/*
不管是GET还是POST方法，访问 xx.com/user 时,都用 XxController 中的 method() 方法去响应 .
*/
Route::match(['get','post'] , '/user' , 'XxController@method')

/*
GET,POST,PUT,DELETE.. 任何方法访问 xx.com/test, 都用第2个参数中的匿名函数去响应 .
*/
Route::any('/test', function () {
    return 'Hello World';
});
```

> 注意: 如果同一个路由被写了2次
> 则以最后一次路由为准!

2.2路由器与传递参数

```
/*
下例是指 xx.com/user/123 这样的 URL,user 后面的值将会捕捉到,
并自动传递给控制器的方法或匿名函数
*/
Route::get('user/{id}', function ($id) {
    return 'User '.$id;
});

/*
下例是指 xx.com/user/{name}/{id} 这样的 URL,user 后的参数,
会被捕捉到 , 并自动传递给控制器的方法或匿名函数
*/
Route::get('user/{name}/{id}', function ($name, $id) {
    return 'user_'.$name.$id;
});
```

如果没有传递参数，则会报错；

2.3 传递可选参数

在路由 参数 的花括号最后 加上 ? (问号) 即可

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

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

2.4 参数限制

在 TP 中,自动验证写在 Model 里,不够灵活. laravel把参数限制写在方法或者路由中.

普通形式：

->where('要限制的参数名','限制规则(正则,不用斜线//)');

数组形式：

->where(['要限制的参数名1'=>'限制规则1(正则,不用斜线//)','要限制的参数名2'=>'限制规则2(正则,不用斜线//)']);

```
Route::get('user/{name}', function ($name) {
    //
})->where('name', '[A-Za-z]+');
Route::get('user/{id}', function ($id) {
    //
})->where('id', '[0-9]+');
Route::get('user/{id}/{name}', function ($id, $name) {
    //
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

注意：路由参数不能包含中横线 "-",参数会被理解为变量名，所以不能有'-'，下划线是可以滴；

第3 章 控制器

3.1 控制器放在哪儿?叫什么 ？

控制器放在'/app/Http/Controllers' 目录下

文件名：XxController.php

例：UserController.php

注意：单词首字母大写 [大驼峰规则]

3.2 控制器类叫什么?命名空间叫什么?继承自谁?

类叫XxController

命名空间是 App\Http\Controllers

继承自App\Http\Controllers\Controller

与用php artisan make:controller UserController的方式的命名空间引入不同

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
```

```
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
class XxxController extends Controller {
    public function add() {
        echo 'hello world';
    }
}
```

也可以放在Controllers的其他目录下

如：App\Http\Controllers\Admin\TestController.php

```
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;

class TestController extends Controller{
    public function index(){
```

```
        return 'Admin\TestController';
    }
}
```

相应路由的写法，如下：

```
Route::get('test','Admin\TestController@index');
```

第4章 模板基础操作

4.1 模板放在哪儿?叫什么?

> 模板放在/resources/view 下.
> 叫什么什么：
> xx.php,或xx.blade.php
> 注意：
> 如果以.php结尾,模板中直接写 PHP 语法即可,例`<?php echo $title; ?>`
> 如果以.blade.php结尾,则可以使用 laravel 特有的模板语法也可以直接使用PHP语法
> 例{{ $title }}
> 如果有 xx.php和xx.blade.php 两个同名模板,优先用 blade 模板.
> 模板中是HTML代码，不要以为是PHP文件就写PHP代码；

4.2 和控制器有什么对应关系?

没有对应关系

直接在控制器方法里引用即可，不像TP一样，有对应关系，不要搞混.
例：

view()可以看成tp中的助手函数。

view在laravel中完成模板渲染，和变量赋值。

```
XxController {
    public function yyMethod(){
        return view('test'); // 将使用 views/test[.blade].php
    }
    public function yyMethod(){
        return view('user.add'); // 将使用 views/user/add[.blade].php
    }
}
```

4.3 模板赋值

将值写到关联数组中，然后将数组写到 view 函数的第二参数中；

```
public function up(){
    $data = ['title'=>'布尔教育','msg'=>'laravel'];
    return view('up',$data);
}
```

一维数组

在view中，直接将数组的键当做变量来使用：

[project]/resources/views/up.php (不带有blade模板，只能用PHP语法)：

```
<h1>
    <?php echo $title;?>
</h1>
<p>
    <?php echo $msg; ?>
</p>
```

[project]/resources/views/up.blade.php(PHP语法和模板语法都支持)：

```
<h1><?php echo $title;?></h1>
<p>
    {{$msg}}
```

```
</p>
```

第5章 数据库迁移

5.1创建数据库

```
create database msg charset usf8
```

5.2修改配置文件

编辑我们项目下的`.env`文件,使之适合自己的服务器环境

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_DATABASE=msg
DB_USERNAME=root
DB_PASSWORD=0000
```

5.3 数据库迁移文件

> 在我们学习创建表时都是`create table Xxx{..`
> 修改表都是 `alert Xxx ...`

> 但是在laravel项目中.是不建议大家使用命令手动建表和修改表的,
> laravel很强大,它把表中的操作写成了migrations迁移文件,
> 然后可以直接通过迁移文件来操作表.
> 所以 , 数据迁移文件就是 **操作表的语句文件**

- 为什么用迁移文件 , 而不直接敲 sql 操作表 ?

1. 便于团队统一操作表.
2. 出了问题,容易追查问题和回溯，有历史回退功能.

迁移文件用命令行生成,不要自己写,生成后再补齐内容;
创建表命令：`php artisan make:migration create_good_table --create=goods`

迁移文件的说明

解释：
artisan
在项目的根目录下，其实就是一个PHP脚本文件，所以用PHP去执行次文件
make:migration
创建迁移文件
create_good_table
自定义文件名--最好能够体现次迁移文件的作用
--create=goods
创建表，表名为goods

执行完命令后，系统会自动创建迁移文件;
在[project]/database/migrations/目录下

我们只需要在 function 中补齐对表的操作即可,比如字段,字段类型等 .
迁移文件是一个类文件
此类中,有2个基本方法,up()和down().
这两个方法,互为逆向操作 .
比如 :
up() 负责建表 , 加列 , 加索引
down() 负责删表 , 减列 , 去索引

5.4 迁移命令的使用

命令行中执行 ：php artisan make:migration create_good_table --create=goods，创建迁移文件;
文件创建成功后，通过修改迁移文件，添加我们需要的相应字段;

```
public function up()
{
    Schema::create('goods', function (Blueprint $table) {
        $table->increments('id');
        $table->char('titles');  //仿照原有的，添加字段
        $table->timestamps();
    });
}
```

修改完成后，执行命令：

```
php artisan migrate
```

运行迁移文件后，查看数据库，就会出现相应的表；
回退命令：php artisan migrate:rollback

不能修改执行后的迁移文件。

想要在表中添加字段，不能修改执行后的迁移文件；
观察迁移文件，是有明确的时间的，再看数据库中的migration表，是有明确记录已经执行过的；

所以我们需要重新生成迁移文件：
修改表命令：
php artisan make:migration add_email_to_good --table=goods
执行完成后，回生成迁移文件，然后修改迁移文件：

```
public function up()
{
    Schema::table('goods', function (Blueprint $table) {
        //添加列
        $table->char('email',50);
    });

 Schema::table('goods', function ($table) {
       //删除列
  $table->dropColumn('email');
 });
}
```

再次执行迁移文件：php artisan migrate ；数据库中就会看到我们新添加的字段；

执行数据库迁移的时候可能会出现以下错误,如

```
[2017-09-25 06:58:52] local.ERROR: SQLSTATE[HY000]: General error: 1709 Index column size too large. The maximum
 column size is 767 bytes. (SQL: alter table `users` add unique `users_email_unique`(`email`))
 {"exception":"[object] (Illuminate\\Database\\QueryException(code: HY000): SQLSTATE[HY000]: General error: 1709
 Index column size too large. The maximum column size is 767 bytes. (SQL: alter table `users` add unique
 `users_email_unique`(`email`)) at D:\\xampp\\htdocs\\mrchi\\laravel\\vendor\\laravel\\framework\\src\\Illuminate\
\Database\\Connection.php:664, PDOException(code: HY000): SQLSTATE[HY000]: General error: 1709 Index column size
 too large. The maximum column size is 767 bytes. at D:\\xampp\\htdocs\\mrchi\\laravel\\vendor\\laravel\\framework\
\src\\Illuminate\\Database\\Connection.php:458)
```

这是由于Laravel 默认使用 utf8mb4 字符，包括支持在数据库存储「表情」。如果你正在运行的 `MySQL release` 版本低于`5.7.7` 或 `MariaDB release` 版本低于`10.2.2` ，为了MySQL为它们创建索引，你可能需要手动配置迁移生成的默认字符串长度，你可以通过调用 `AppServiceProvider` 中的 `Schema::defaultStringLength` 方法来配置它：

```
use Illuminate\Support\Facades\Schema;

/**
 * 引导任何应用程序服务。
 *
 * @return void
 */
public function boot()
```

```
{
    Schema::defaultStringLength(191);
}
```

5.5 数据库迁移操作

当迁移文件做好的之后，以下几个命令，执行迁移文件 .

`php artisan migrate` 执行所有迁移文件

`php artisan migrate:rollback` 回退到最近执行迁移的状态

`php artisan migrate:reset` 回退到所有迁移之前的初始状态

`php artisan migrate:refresh` 回退到初始状态，再次执行所有迁移文件

`php artisan migrate:fresh` 删除数据表,再次执行所有迁移文件(5.5新增)

`php artisan migrate:install`重置并重新运行所有的 migrations

`php artisan migrate --force:` 强制执行最新的迁移文件

重置的是migrations表
而不是迁移文件

5.5 迁移语法速查表

可用的字段类型:

结构构造器包含了许多字段类型，供你构建数据表时使用：

数据库结构构造器包含了许多字段类型，供你构建数据表时使用：

命令	描述
`$table->bigIncrements('id');`	递增 ID（主键），相当于「UNSIGNED BIG INTEGER」型态。
`$table->bigInteger('votes');`	相当于 BIGINT 型态。
`$table->binary('data');`	相当于 BLOB 型态。
`$table->boolean('confirmed');`	相当于 BOOLEAN 型态。
`$table->char('name', 4);`	相当于 CHAR 型态，并带有长度。
`$table->date('created_at');`	相当于 DATE 型态
`$table->dateTime('created_at');`	相当于 DATETIME 型态。
`$table->dateTimeTz('created_at');`	DATETIME (带时区) 形态
`$table->decimal('amount', 5, 2);`	相当于 DECIMAL 型态，并带有精度与基数。
`$table->double('column', 15, 8);`	相当于 DOUBLE 型态，总共有 15 位数，在小数点后面有 8 位数。
`$table->enum('choices', ['foo', 'bar']);`	相当于 ENUM 型态。
`$table->float('amount', 8, 2);`	相当于 FLOAT 型态，总共有 8 位数，在小数点后面有 2 位数。
`$table->increments('id');`	递增的 ID (主键)，使用相当于「UNSIGNED INTEGER」的型态。
`$table->integer('votes');`	相当于 INTEGER 型态。
`$table->ipAddress('visitor');`	相当于 IP 地址形态。
`$table->json('options');`	相当于 JSON 型态。
`$table->jsonb('options');`	相当于 JSONB 型态。
`$table->longText('description');`	相当于 LONGTEXT 型态。
`$table->macAddress('device');`	相当于 MAC 地址形态。
`$table->mediumIncrements('id');`	递增 ID (主键)，相当于「UNSIGNED MEDIUM INTEGER」型态。

`$table->morphs('taggable');`	加入整数 `taggable_id` 与字符串 `taggable_type`。
`$table->nullableMorphs('taggable');`	与 `morphs()` 字段相同，但允许为NULL。
`$table->nullableTimestamps();`	与 `timestamps()` 相同，但允许为 NULL。
`$table->rememberToken();`	加入 `remember_token` 并使用 VARCHAR(100) NULL。
`$table->smallIncrements('id');`	递增 ID (主键)，相当于「UNSIGNED SMALL INTEGER」型态。
`$table->smallInteger('votes');`	相当于 SMALLINT 型态。
`$table->softDeletes();`	加入 `deleted_at` 字段用于软删除操作。
`$table->string('email');`	相当于 VARCHAR 型态。
`$table->string('name', 100);`	相当于 VARCHAR 型态，并带有长度。
`$table->text('description');`	相当于 TEXT 型态。
`$table->time('sunrise');`	相当于 TIME 型态。
`$table->timeTz('sunrise');`	相当于 TIME (带时区) 形态。
`$table->tinyInteger('numbers');`	相当于 TINYINT 型态。
`$table->timestamp('added_on');`	相当于 TIMESTAMP 型态。
`$table->timestampTz('added_on');`	相当于 TIMESTAMP (带时区) 形态。
`$table->timestamps();`	加入 `created_at` 和 `updated_at` 字段。
`$table->timestampsTz();`	加入 `created_at` and `updated_at` (带时区) 字段，并允许为NULL。
`$table->unsignedBigInteger('votes');`	相当于 Unsigned BIGINT 型态。
`$table->unsignedInteger('votes');`	相当于 Unsigned INT 型态。
`$table->unsignedMediumInteger('votes');`	相当于 Unsigned MEDIUMINT 型态。
`$table->unsignedSmallInteger('votes');`	相当于 Unsigned SMALLINT 型态。
`$table->unsignedTinyInteger('votes');`	相当于 Unsigned TINYINT 型态。
`$table->uuid('id');`	相当于 UUID 型态。

字段修饰

除了上述的字段类型列表，还有一些其它的字段「修饰」，你可以将它增加到字段中。例如，若要让字段「nullable」，那么你可以使用 `nullable` 方法：

```
Schema::table('users', function (Blueprint $table) {
    $table->string('email')->nullable();
});
```

以下列表为字段的可用修饰。此列表不包括 索引修饰：

Modifier	Description
`->after('column')`	将此字段放置在其它字段「之后」（仅限 MySQL）
`->comment('my comment')`	增加注释
`->default($value)`	为此字段指定「默认」值
`->first()`	将此字段放置在数据表的「首位」（仅限 MySQL）
`->nullable()`	此字段允许写入 NULL 值
`->storedAs($expression)`	创建一个存储的生成字段（仅限 MySQL）
`->unsigned()`	设置 `integer` 字段为 `UNSIGNED`
`->virtualAs($expression)`	创建一个虚拟的生成字段（仅限 MySQL）