

Laravel 入门教程

- ## php命令行的调用

- 执行php文件时，浏览器通过apache的80端口，apache告诉php解释器，php.exe返回结果给apache，apache返回浏览器，显示页面。

-

- 设置php.exe 快速启动方式 path设置环境变量，然后在cmd里面可以直接使用： php 文件名

-

- ## PHP类库管理器

- 在之前，类库很多

- 1.没有统一的资源仓库，到处乱找
- 2.没有统一的安装方式，rar、zip、tar各种包都有，下载后自己得整理
- 3.遇到库的依赖关系，得自己再次下载解决

- composer packagist.org

- 其他语言也有类似工具：java有maven,python有pip,node.s有npm,前端有bower

-

- ## 安装composer

- 要求：

- php >= 5.5

- Openssl扩展 extension=php_openssl.dll

- PDO扩展 extension=php_pdo_mysql.dll

- Mbstring扩展

-

- 下载composer，然后安装。

-

- 离线安装，因为墙，下载扩展包的时候，会慢，可以改用这种方式。
 - 下载好离线包，放在php解释器下目录下。然后就可以了
 - composer -v 检验下是否有安装上
 - 配置composer修改为国内镜像
 - composer config -g 镜像

-

- ## 初试composer

- 为项目引入某个库

- 在项目里面写个composer.json

- {

- }

- "厂商/类库":版本号

- 然后再项目里，开启cmd，使用命令 composer install

-
- 如何加载引入的库
- `require(__DIR__.'/vendor/autoload.php');`
-
- 添加某个新库
- ```
{
 • "require":{
 • "smarty/smarty":"3.1.31",
 • "phpmailer/phpmailer":"3.1.13",
 • }
```
- }
- 这时候composer install 可能会报错。因为composer.lock会报错
- `composer update`
- 
- 卸载类库
- `composer remove phpmailer`
- 

## • 不配置composer.json安装

- `composer require phpmailer//phpmailer=版本号`
- 

## • composer创建项目

- `composer create-project laravel/laravel=5.7.19`
- 

## • 第2章 路由器

- 1.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
- 
- laravel的路由器与控制器的关系，需要明确的在<project>/app/Http/routes.php文件中明确定义
- 同样的路由，以第二个去响应
- 
- 基础路由：
- ```
Route::get('/', function () {
    • return view('welcome');
```
- });
-
- 多请求路由
- `Route::get('admin/login',function(){});`
- `Route::post('admin/login',function(){});`
- 这样太复杂了，太乱了

- `Route::match(['get','post'],'admin/get',function(){`
 - `return 'login';`
- `}); //自己匹配`
-
- `Route::any('admin/register',function(){`
 - `return 'register'`
- `}); //匹配任何`
-

路由传参

- `Route::get('Home/user/{$id}', function($id){`
 - `return 'user'.$id;`
- `});`
- 多个参数
- `Route::get('Home/user/{id}/{name}',function($id,$name){`
 - `return 'user'.$id.$name;`
- `});`
- `//传递可选参数`
- `Route::get('goods/{page?}',function($page = 1){`
 - `return 'page'.$page;`
- `});`
-

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($name){`
- `return 'user'.$name;`
- `})->where('name','[A-Za-z]*');`
-

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

模版操作

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?

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:

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

创建数据库

数据库迁移

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。

比如你在自己电脑上create table xqx(),建了一张表。
但其他几个程序员,如何和你保持同步?也打开 mysql 控制台执行一遍?
都执行一遍当然可以,但很容易各程序员操作不一致的情况。
把操作数据库的语句,写在文件里,大家用同一份文件操作表,就能保持高度一致了。
其实就是把你表的操作,都体现在文件上,而不是随手敲个命令改表。
假设出现不一致的情况,也有历史记录可以回退:

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`

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去执行该文件
make:migration
创建迁移文件
create_good_table
自定义文件名--最好能够体现该迁移文件的作用
--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

- 使用 `php artisan migrate` 的时候出现以下提示?
- Migration table created successfully.
- [Symfony\Component\Debug\Exception\FatalThrowableError]
- Call to undefined method Illuminate\Database\Schema\Blueprint::test()
-
- 字段写错,应该是text,写成test,如果有相应的错误,可以检查是否哪个字段的属性写错了。

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:

- 添加列的话,有写添加列,也要写删除列,不然回退的话,会没法生效。

```

    public function up()
    {
        Schema::table('goods', function (Blueprint $table) {
            //
            $table->string('email');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('goods', function (Blueprint $table) {
            //
            $table->dropColumn('email');
        });
    }
}

```

再次执行迁移文件：php artisan migrate 数据库中将会有我们新添加的字段！

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: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

- 用artisan创建控制器
- php artisan make:controller MyController
- 需要在某个目录下创建的话
- php artisan make:controller Admin\MyController

• 迁移文件速查表

- <https://laravel-china.org/docs/laravel/5.5/migrations/1329#b419dd>
- 在可用字段类型下。

• 第6章 DB类操作数据库

第6章 DB类操作数据库

按 MVC 的架构,数据库的操作大部分应放在 Model 中,

但如果不用 Model,我们也可以用 laravel 的 DB 类操作数据库.

而且,如果某些极其复杂的sql,用 Model 已经很难表达,要手写sql.也需要用 DB 类去执行原生sql.
laravel 中 DB 类的基本用法:DB::table('users') 获取操作users表的实例.

6.1 insert 添加操作

插入单行,一维数组形式,数组的键就是表的字段,返回值为true 和 false;

```
$row = ['titles'=>'哈哈','email'=>'test@qq.com'];  
DB::table('goods')->insert($row);
```

插入多行 (多维数组)

```
$rows = array(  
    array('titles'=>'哈哈111','email'=>'lisi@qq.com'),  
    array('titles'=>'哈哈222','email'=>'wang@qq.com')  
);  
DB::table('goods')->insert($rows);
```

- 添加多行时候, insertGetId 不能使用.

插入后返回主键值 获取主键值,用insertGetId()方法,(多维数组不行??)

```
$rows = array('titles'=>'哈sdak','email'=>'wan12g@yy.com');  
$id = DB::table('goods')->insertGetId($rows);  
var_dump($id);
```

6.2 update 修改操作

- 典型修改

```
DB::table('users')->where('id', 1)->update(['age' => 19])
```

相当于sql:

```
update users set age=19 where id=1 ;
```

- 某字段在原基础上 增长或减少 increment/decrement
返回值是受影响的行数;

```
DB::table('users')->where('id',1)->increment('age');//默认步长为1  
DB::table('users')->where('id',2)->increment('age', 3); //第二个参数,指定步长  
DB::table('users')->where('id',3)->decrement('age');  
DB::table('users')->where('id',4)->decrement('age', 3);
```

6.3 delete 删除操作

```
var_dump(DB::table('goods')->where('id', '>', 3)->delete());  
//where 有三个参数时，其中第二个参数当做运算符  
//返回受影响的行数
```

6.4 查找操作

注意：取出的数据，无论是单行还是多行，每一行数据都是以一个对象的形式组织的，不是关联数组。

```
// select * from users;  
DB::table('goods')->get();  
// select * from user where id > 6  
DB::table('goods')->where('id', '>' 6)->get();  
// select id,email from users where id > 6  
DB::table('goods')->select('id','email')->where('id', '>' 6)->get();  
// select * from users where id=6 取出单行，返回  
DB::table('goods')->where('id',6)->first()
```


第8章 blade模板

laravel 有自己的模板引擎,以.blade.php结尾.

语法相较TP模板和Smarty模板更简洁一些.

8.1 数据要集中传递到模板

在 Smarty 和 TP 模板中,要把变量assign 给模板引擎.

例:

```
$smarty->assign('title'=>' 今天天气不错 ');
$smarty->assign('content'=>' 温度零上 13 度 ');
```

在 blade 模板中,不是assign, 而是以数组参数集中传递.

例:

```
$data = [
    'title'=>' 天气预报 ',
    'content'=>' 今天天气真不错 ',
    'score'=>mt_rand(40,90),
    'score'=>mt_rand(40,90),
    'users'=>['zhangsan','lisi','wangwu']
];
return view('test',$data);|
```

模板中, 普通变量的使用: {{ \$title }} ==> 天气预报

8.2 模板判断

```
public function test(){
    $arr = ['ti'=>'13','de'=>'sldak','user'=>['1','3','55']];
    return view('msg.test',$arr);
}
```

```
@if (express) # 注意 express 两边加括
@elseif (express) # 表达示中
@else
@endif
```

例：

```
{{ $score }}
@if ($score >= 80)
    优秀
@elseif ($score >= 60)
    及格
```

例：

```
{{ $score }}
@if ($score >= 80)
    优秀
@elseif ($score >= 60)
    及格
@else
    不及格
@endif
```

除非,和 if 相反:

```
@unless ($score >= 60)
    除非 score 大于等于60, 否则显示不及格
@endunless
```

8.3 循环

for循环:

```
@for ($i=0; $i<10; $i++)  
    {{ $i }} <br>  
@endfor
```

foreach 循环:

```
@foreach ($user as $u)  
    {{ $u }}  
@endforeach
```

ifelse 循环是否为空

```
@forelse ([ ] as $u)  
    {{ $u }} //如果数组有数据显示数据  
@empty  
    nobody //如果数组为空, 则显示  
@endforelse
```

8.4 模板包含与继承

包含:

@include('msg.sub') 包含views 下的msg/sub.blade.php

继承: I

模板继承比模板包含更强大.

如下, 一个典型的网页结构

头部和尾部都一样, 就中间的左右内容不一样.

8.5 不解析模板和防xss攻击

在一些前端模板引擎中,也有可能用`{{}}`做标签边界,为防止blade模板去解析,前面加`@`符号阻止解析.

例: `@{{jsvar}}`

防 XSS 攻击:

```
['code'=>'<script>alert(1)</script>']  
输出到 view 层,看源码:  
&lt;script&gt;alert(1)&lt;/script&gt;  
如果确实不需要实体转义,可以在变量两边加!! (1个大括号,不是两个):  
例: {!!$code!!}
```

第9章 强大的 Model

9.1 Model放在哪儿?命名空间是什么?

model 文件默认放在/app目录下,命名空间是App.

model 文件也可以自由的放在其他目录,但请注意命名空间和目录路径保持一致.

9.2 Model类叫什么?继承自谁?

在laravel中约定(非强制),表名叫xs,复数形式.

如用户(user)表名叫users,邮件(email)表叫emails.

类和表名有关系,一般表名去掉s,即为Model的类名.

所以:

users表的Model类叫class User.

emails表的Model类叫class Email,注意首字母大写.

继承自

```
Illuminate\Database\Eloquent\Model
```

9.3 自动生成和实例化

Model 可以手写, 也可以用 artisan 命令行工具生成.

例: `php artisan make:model Msg`

实例化:

```
$model = new App\Xxx(); // 得到 Xxxs 表的 Model, 且不与表中任何行对应 .  
$model = App\Xxx::find(4); // 静态方法调用, 得到 Xxxs 表的 Model, 且与 $id=4 的数据对应 .
```

9.5 增

```
public function add() {  
    $msg = new App\Msg();  
    $msg->title = $_POST['title'];  
    $msg->content = $_POST['content'];  
    return $msg->save() ? 'OK' : 'fail';  
}
```

9.6 查

查单行: `find()` 与 `first()`

```
// 按主键查  
Msg::find($id) // 按主键查  
// 按 where 条件查具体那一条  
Msg::where('id', '>', 3)->first();
```

查多行: `all()` 和 `get()`

```
// 无条件查所有行 . select 列 1, 列 2 from msgs;  
Msg::all(['列1', '列2']); // 按条件查多行  
  
// 按条件查多行  
Msg::where('id', '>', 2)->get(['列 1', '列 2']); // 数组选列  
Msg::where('id', '>', 2)->select('title', 'content')->get(); // 字符串选列
```

9.7 改

```
public function up($id) {  
    if( empty($_POST) ) {  
        $msg = Msg::find($id);  
        return view('msg.up', ['msg'=>$msg]);  
    }else {  
        $msg = Msg::find($id);  
        $msg->title = $_POST['title'];  
        $msg->content= $_POST['content'];  

```

```
        return $msg->save() ? 'OK' : 'fail';  
    }  
}
```

9.8 删

先找到，然后删

```
public function del($id) {  
    $msg = Msg::find($id);  
    return $msg->delete() ? 'ok' : 'fail';  
}
```

用条件选择直接删：

```
public function del($id) {  
    return Msg::where('id',$id)->delete()? 'ok' : 'fail';  
}
```

9.9 复杂查询

排序：

```
// select ... where id > 2 order by id desc;  
Msg::where('id','>',2)->orderBy('id','desc')->get();
```

**** 限制条目 ****

```
// select .. where id>2 order by id desc limit 2,1;  
//跳过两行 取一行  
Msg::where('id','>',2)->orderBy('id','desc')->skip(2)->take(1)->get();
```

统计：

```
Msg::count();  
Msg::avg('id');  
Msg::min('id');  
Msg::max('id');  
Msg::sum('id');
```

更加复杂的查询：

!(<http://laravel-china.org/docs/5.1/queries>)[<http://laravel-china.org/docs/5.1/queries>]

9.10 你和model有个约定

- 表名的约定

默认表名为Model名 + s,如果不想这样做,可以通过的 model 类的table 属性来指定表名。
例:

```
class XxModel extends Model {  
    protected $table = 'yourTableName';  
}
```

- id 的约定

Model 默认 认为,每张表都有一个叫做id的主键,
如果向通过其他字段名来设置主键,可以通过primaryKey属性来指定主键列名

```
class XxModel extends Model {  
    protected $primaryKey = 'Xx_id'; //注意: Key 首字母大写  
}
```


图片加载中...



图片加载中...



图片加载中...



图片加载中...



图片加载中...



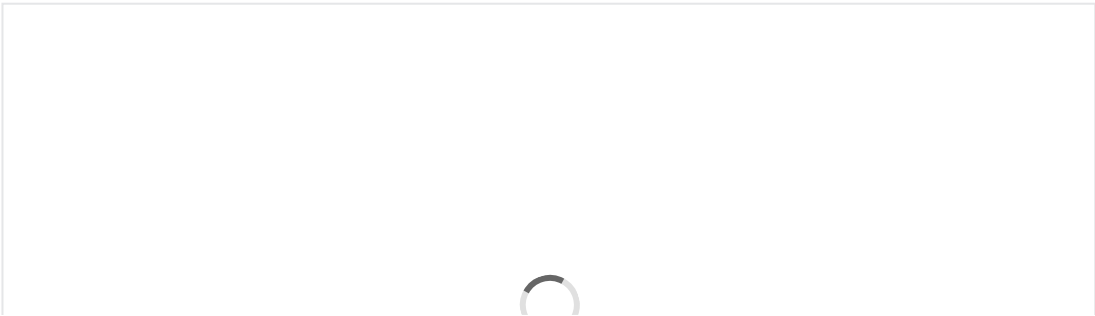
•



•



•





图片加载中...



图片加载中...



图片加载中...





图片加载中...



图片加载中...



图片加载中...