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
- {
    - "requeire":{
        - "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 reuqire 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/Htpp/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 xxx(),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,就能操持高度一致了.
其实就是把你对表的操作,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

- 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。
-
-

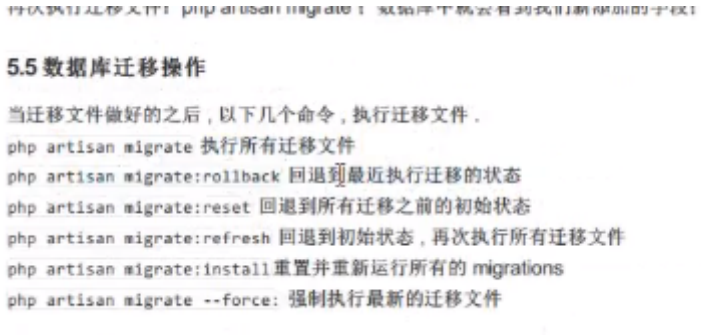
想要在表中添加字段,不能修改执行后的迁移文件;
观察迁移文件,是有明确的时间的,再看数据库中的migration表,是有明确记录已经执行过的;

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。
-

```php
    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');
        });
    }
```



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

- 添加多行时候，insertGetId 不能使用。
-

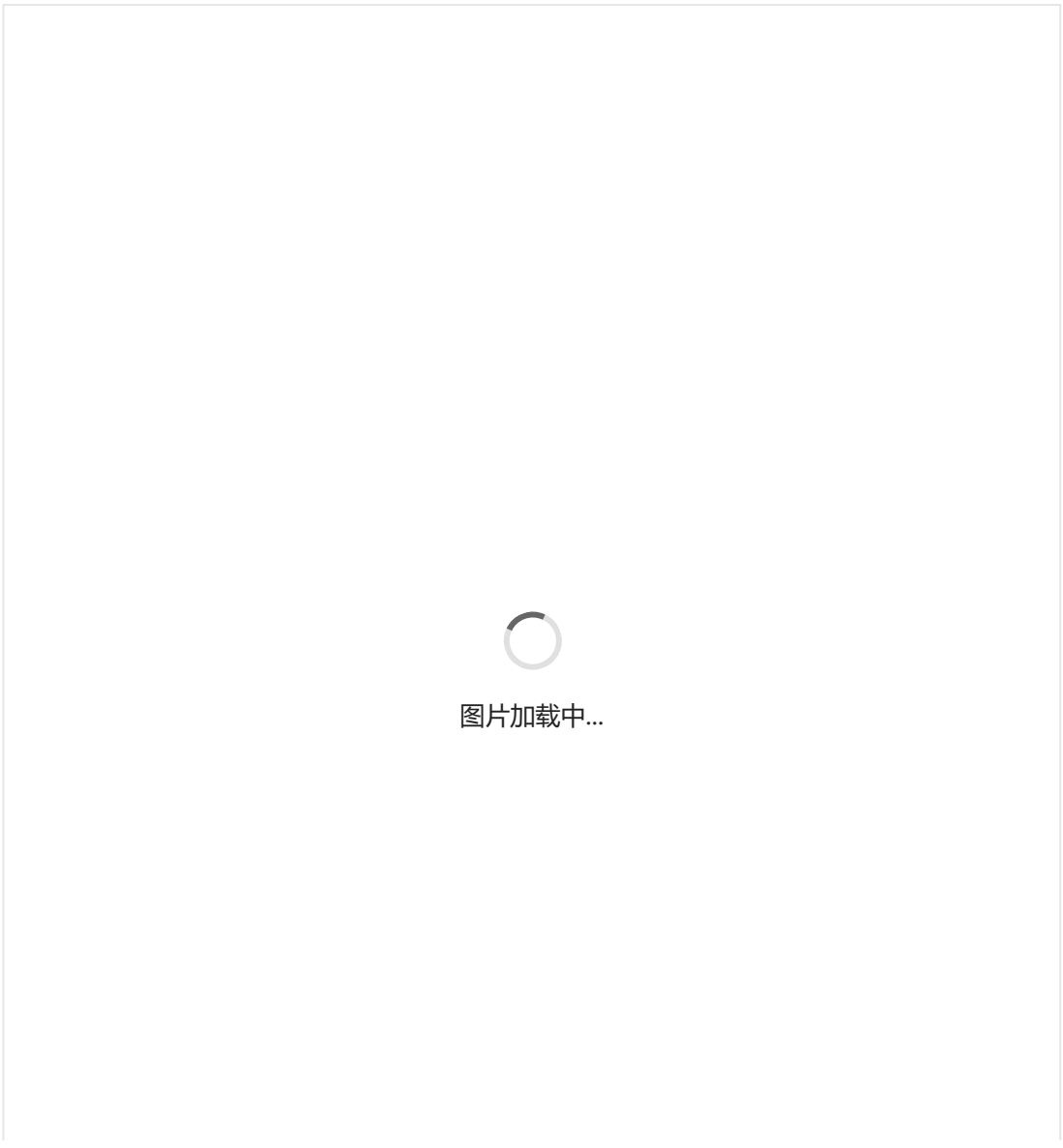

-
-


-


-
-


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...