laravel -- 艺术家的框架

1章 如何安装laravel

1.1 laravel目录结构

laravel再牛,也是PHP写的一堆代码.
他的大致目录如下:

```
/laravel
    /app
    /artisan
    /bootstrap
    /composer.json
    /config
    /database
    /gulpfile.js
    /package.json
    /phpunit.xml
    /public
    /readme.md
    /resources
    /server.php
    /storage
    /tests
    /vendor
```

laravel自身有很多PHP文件,同时有个`composer.json`文件.
说明他需要依赖很多库。

1.2 四种安装办法

我们该如何安装laravel,网上你大概能找到4种方法,但你不要去死记.
而是要从原理推出:
`完整的laravel = laravel本身 + composer指定的依赖库`

所以你至少可以有这4种办法:

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=5.1.1
```

1. 下载别人帮我拼装好的`laravel本身+composer中指定的库`
   雷锋在这: http://www.golaravel.com/download/
2. 手动下载laravle本身,composer安装依赖库(何苦半自动?)
   https://github.com/laravel/laravel/tree/5.1
   wget 下载laravel,再cd到目录下,执行`composer install`
3. `larvel安装器`,可以帮你完成这两步(不推荐)

```
# 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>
```

1.3 修改laravel 目录权限

在linux下使用laravel,需要修改目录的权限

```
chmod o+rwx <project>/storage -R
chmod o+rwx <project>/bootstrap/cache -R
```

1.4 配置虚拟主机

在nginx中,配置server段,指向`<project>/public`目录.
依我们要做的p2p金融网站为例:

```
location / {
            root   html/<proected>/public;
            index  index.php index.html index.htm;
        }
```

在apache中,

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

1.5 配置简洁URL

对于apache,需要开启`rewrite`重写模块,
在`http.conf`中,把下一行前`#`去掉
`#LoadModule rewrite_module modules/mod_rewrite.so`

在`<project>/public`创建`.htaccess`文件,内容如下:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

如果仍不可以,找如下代码,把`None`改为`All`

```
# AllowOverride controls what directives may be placed in .htaccess files.
# It can be "All", "None", or any combination of the keywords:
#   AllowOverride FileInfo AuthConfig Limit
#
AllowOverride None
```

对于nginx,在虚拟主机配置段中加如下一句

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

nginx实例:

```
location / {
            root   html/jinrong/public;
            try_files $uri $uri/ /index.php?$query_string;
            index  index.php index.html index.htm;
        }
```

1.6 可能出现的异常及解决

** 1 Whoops, looks like something went wrong.**
如果页面只有这一行错误,这是因为配置文件没有生成.
laravel的配置文件叫'.env';
`cp .env.example .env`

2. No supported encrypter found

```
RuntimeException in EncryptionServiceProvider.php line 29:
No supported encrypter found. The cipher and / or key length are invalid.
```

原因:laravel需要定义一个key,用于作密钥用,但没生成这个key.
解决:`php artsian key:generate` 生成key

2. 服务器500
这种情况下,首先确保php环境是正确的,虚拟主机配置也正常.
然后问题应该出现在目录的权限上. 请按1.3章节继续操作.

2章 路由器

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>/app/Http/routes.php`
文件中明确定义.

格式如下:

```
/*
下例指: 当用GET方式访问 xx.com/yy 这个地址时,用XxController中的reg()方法去响应.
*/
Route::get('/yy', 'XxController@reg');

/*
当用POST方式访问 xx.com/zz 这个地址时,用XxController中的pay()方法去响应.
*/
Route::post('/zz', 'XxController@pay');

/*
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';
})

/*
GET/POST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/foo/bar,都用第2个参数中的匿名函数去响应.
*/
Route::any('/foo/bar', 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 路由器与参数传递

```
/*
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/cat/{cat}/page/{page}这样的URL,cat后的参数和page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('cat/{cat}/page/{page}', function ($catid, $pageid) {
    //
});
```

2.3 传递可选参数

```
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里.

```
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"-",(想想变量名规范)
> 可以用下划线"_".

3章 控制器

3.1 控制器放在哪儿?

控制器放在`'<project>/app/Http/Controllers'`目录下

3.2 控制器文件叫什么?

```
文件名: Xx控制器Controller.php
例: UserController.php
注意: 单词首字母大写[大驼峰规则]
```

3.3 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`

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

class XxxController extends Controller {
  public function add() {
  }
}
```

4章 模板操作

4.1 模板放在哪儿?

模板放在`<project>/resources/view`下.

4.2 叫什么?

`xx.php`,或`xx.blade.php`
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,则是用laravel特有的模板语法.例`{{ $title }}`

> 注意: 如果有 `xx.php`, `xx.blade.php`两个同名模板, 优先用blade模板.

4.3 和控制器有什么对应关系?

直接在控制器方法里引用即可.
例:

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

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

5章 数据库迁移

5.1 创建数据库

`create database lion charset utf8`

5.2 修改参数

编辑项目下的.env文件,使之适合自己的服务器环境

```
DB_HOST=localhost
DB_DATABASE=lion
DB_USERNAME=root
DB_PASSWORD=
```

5.3 数据迁移文件

依我们目前的知识,需要建表`create table xxx(...)`,
需要改表`alter table xxx...`.

但是,在laravel中,不建议用命令手工建表和修改表,
而是把对表的操作写成migration迁移文件.
然后lavavel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.
       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DDL语言写在文件里,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.

    2. 出了问题,容易追查责任和回溯.
       上次我们的微信平台,从2/29号新关注的用户信息不能入库.
       我翻历史命令时,发现有人执行`alter table`语句,而且该语句把表的`auto_increment`属性去掉了,导致了主键重复,无法入库.

而此人,正是徐振东助教!
而且他意识到错误后,把表结构又改了回去,装做没事儿人.
经审讯后,助教承认他修改过表.

如果有表迁移文件,责任就好找多了.
而且,即使出错,只需要用`migration`工具回退表结构即可.

- 迁移文件用命令行生成,自己再补齐内容 两例: `php artisan make:migration create_good_table --create=goods`
  `php artisan make:migration add_price_to_good --table=goods`

看到如下两个文件:
database/migrations/2016_01_18_083644_goods.php:

```
class Goods extends Migration
```

```
{
    public function up()
    {
        Schema::create('goods', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('goods');
    }
}
```

database/migrations/2016_01_18_083650_goods.php

```
class goods extends Migration {

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

    public function down()
    {
        Schema::table('goods', function (Blueprint $table) {
            //
        });
    }
}
```

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

- 迁移文件分 **创建表** 和 **修改表** 两种
  在上一节中, 仔细观察2个类的方法.

`--create=xx表`,则该类的`up()`中调用`Schema::create()`,即创建表.
`--table=xx表`,则该类的`up()`中调用`Schema::table()`,即修改表.

- 迁移文件的名字以能说明文件的作用最好. 例: `create_goods_table`, 创建商品表 `add_age_to_stu_table`, 为学员表添加age字段

5.4 数据迁移操作

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
- php artisan migrate:rollback
- php artisan migrate:reset
- php artisan migrate:refresh
- php artisan migrate:status
- php artisan migrate:install

migrate: 执行所有迁移文件
migrate --force: 强制执行最新的迁移文件
migrate:rollback ** 回退到执行迁移前的状态
****migrate:reset** 回退到所有迁移之前的初始状态**
migrate:refresh 回退到初始状态,再次执行所有迁移文件

5.5 迁移语法速查表

https://laravel.com/docs/5.1/migrations

常见列类型一览:

Command	Description
`$table->bigIncrements('id');`	Incrementing ID (primary key) using a "UNSIGNED BIG INTEGER" equivalent.
`$table->bigInteger('votes');`	BIGINT equivalent for the database.
`$table->binary('data');`	BLOB equivalent for the database.
`$table->boolean('confirmed');`	BOOLEAN equivalent for the database.
`$table->char('name', 4);`	CHAR equivalent with a length.
`$table->date('created_at');`	DATE equivalent for the database.
`$table->dateTime('created_at');`	DATETIME equivalent for the database.
`$table->decimal('amount', 5, 2);`	DECIMAL equivalent with a precision and scale.
`$table->double('column', 15, 8);`	DOUBLE equivalent with precision, 15 digits in total and 8 after the decimal point.
`$table->enum('choices', ['foo', 'bar']);`	ENUM equivalent for the database.
`$table->float('amount');`	FLOAT equivalent for the database.
`$table->increments('id');`	Incrementing ID (primary key) using a "UNSIGNED INTEGER" equivalent.
`$table->integer('votes');`	INTEGER equivalent for the database.
`$table->json('options');`	JSON equivalent for the database.
`$table->jsonb('options');`	JSONB equivalent for the database.
`$table->longText('description');`	LONGTEXT equivalent for the database.
`$table->mediumInteger('numbers');`	MEDIUMINT equivalent for the database.
`$table->mediumText('description');`	MEDIUMTEXT equivalent for the database.
`$table->morphs('taggable');`	Adds INTEGER `taggable_id` and STRING `taggable_type`.
`$table->nullableTimestamps();`	Same as `timestamps()`, except allows NULLs.
`$table->rememberToken();`	Adds `remember_token` as VARCHAR(100) NULL.
`$table->smallInteger('votes');`	SMALLINT equivalent for the database.
`$table->softDeletes();`	Adds `deleted_at` column for soft deletes.
`$table->string('email');`	VARCHAR equivalent column.
`$table->string('name', 100);`	VARCHAR equivalent with a length.
`$table->text('description');`	TEXT equivalent for the database.
`$table->time('sunrise');`	TIME equivalent for the database.
`$table->tinyInteger('numbers');`	TINYINT equivalent for the database.
`$table->timestamp('added_on');`	TIMESTAMP equivalent for the database.
`$table->timestamps();`	Adds `created_at` and `updated_at` columns.
`$table->uuid('id');`	UUID equivalent for the database.

常用列修改方法:

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

Modifier	Description
`->first()`	Place the column "first" in the table (MySQL Only)
`->after('column')`	Place the column "after" another column (MySQL Only)
`->nullable()`	Allow NULL values to be inserted into the column
`->default($value)`	Specify a "default" value for the column
`->unsigned()`	Set `integer` columns to `UNSIGNED`

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.

DB类的基本用法: `DB::table('users')` 获取操作`users`表的实例.

6.1 增 insert

- 插入单行 (注意看数组的键)

```
DB::table('users')->insert(['email'=>'lisi@qq.com']);
```

`insert()`方法返回值为`true` 和 `false`

- 插入多行 (多维数组)

```
DB::table('users')->insert([
                            ['email'=>'lisi@qq.com'],
                            ['email'=>'wang@yy.com'],
                        ]);
```

- 插入后返回主键值 获取主键值,用`insertGetId()`方法

```
DB::table('users')->insertGetId(['email'=>'lisi@qq.com']);
```

6.2 改 update

- 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`;
- 某字段增长或减少 incr/decr

```
DB::table('users')->where('id',1)->increment('age');
DB::table('users')->where('id',2)->increment('age', 3);
DB::table('users')->where('id',3)->decrement('age');
DB::table('users')->where('id',4)->decrement('age', 3);
```

6.3 删 delete

```
DB::table('users')->where('id' , '>' , '6')->delete();
```

6.4 查 select

```
// select * from users;
DB::table('users')->get();
```

```
// select * from user where id > 6
DB::table('users')->where('id' , > 6)->get();

// select id,email from users where id > 6
DB::table('users')->select('id','email')->where('id' , > 6)->get();

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

> 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.

7章 完整的增删改查

7.1 程序规划

- GET /msg/index 展示留言列表
- GET /msg/add 展示表单
- POST /msg/add 接受POST数据,并入库
- GET /msg/del/{id} 删除留言
- [GET,POST] /msg/up/{id} 修改留言

按规划写如下路由器

```
Route::get('msg/index' , 'MsgController@index');
Route::get('msg/add' , 'MsgController@add');
Route::post('msg/add' , 'MsgController@addPost');
Route::get('msg/del/{id}' , 'MsgController@del');
Route::match(['get','post'],'msg/up/{id}' , 'MsgController@up');
```

7.2 数据迁移

1. 生成迁移文件
   `php artisan make:migration create_msgs_table --create=msgs`
2. 编辑迁移文件

```
public function up() {
    Schema::create('msgs', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title',50);
        $table->string('content',200);
        $table->integer('pubtime');
        $table->timestamps();
    });
}
public function down() {
    Schema::drop('msgs');
}
```

1. 执行迁移 `php artisan migrate`

7.3 发布留言

表单页

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

class MsgController extends Controller {
  public function add() {
    return view('msg.add');
  }
}
```

```
// template <project>/resource/views/msg/add.php
<html>
<meta charset="utf-8">
<body>
<h1>laravel添加留言</h1>
<form action="" method="post">
  <p><input type="text" name="title"></p>
  <p>
    <textarea name="content"></textarea>
  </p>
  <p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

发布页:

```
public function post() {
    $rs = DB::table('msgs')->insert(['title'=>$_POST['title'] ,
        'content'=>$_POST['content']]);
    return $rs ? 'OK' : 'fail';
  }
```

提交出错:`TokenMismatchException in VerifyCsrfToken.php line 53:`
不要惊慌,这是因为laravel自带防站外提交(Csrf)的功能.
原理:加入某个特征串,在POST接收页面检测此特征串.
解决: 在表单中,加入这个特征串就行了.
`<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">`

列表页:

```
public function index() {
  $msgs = DB::table('msgs')->get();
  return view('msg.index' , ['msgs'=>$msgs]);
}
```

```
<html>
<body>
  <h1>所有留言</h1>
  <table>
    <tr>
      <td>标题</td>
      <td>内容</td>
      <td>操作</td>
    </tr>
    <tr>
    <?php foreach($msgs as $m) { ?>
       <td><?php echo $m->title;?> </td>
       <td><?php echo $m->content;?></td>
       <td><a href="/msg/del/<?php echo $m->id;?>">删除</a>
       |
       <a href="/msg/up/<?php echo $m->id;?>">修改</a></td>
    <?php } ?>
    </tr>
  </table>
</body>
</html>
```

删除页+跳转功能

```
public function del($id) {
     if( DB::table('msgs')->where('id',$id)->delete() ) {
       return redirect('msg/index');
     } else {
       return 'del error';
     }
  }
```

修改页

```
<html>
<meta charset="utf-8">
<body>
<h1>laravel修改留言</h1>
<form action="" method="post">
  <p><input type="text" name="title" value="<?php echo $msg->title;?>"></p>
  <p>
    <textarea name="content"><?php echo $msg->content;?></textarea>
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
  </p>
  <p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

```
public function up($id) {
  if( empty($_POST) ) {
    $msg = DB::table('msgs')->where('id',$id)->first();;
    return view('msg.up',['msg'=>$msg]);
  }else {
    $rs = DB::table('msgs')->where('id',$id)->update(
     ['title'=>$_POST['title'],
     'content'=>$_POST['content'],
    ]);
    return $rs ? 'OK' : 'false';
  }
}
```

> 至此,我们已经用laravel做了一个简单留言板
> 从增删改查的角度讲,此时你可以用laravel做任意网站了.
> 但是,laravel还有很多漂亮的功能没有用上
> 接下来,继续深入学习laravel

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:

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

`view('xx' , $data);`
```

8.2 普通变量

`{{$title}}` -> `天气预报`

8.3 if/else

```
@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
@else
不及格
@endif
```

8.4 unless (除非,和if相反)

```
@unless ($score >= 60)
  不及格
@endunless
```

8.5 for循环

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

8.6 foreach循环

```
@foreach ($users as $u)
   {{$u}} <br>
@endforeach
```

8.7 forelse 循环是否为空

```
@forelse ([] as $u)
  {{$u}}
@empty
  nobody
@endforelse
```

8.8 模板包含

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

8.9 模板继承

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.

```
________
|_______|
|    |  |
|    |  |
|____|__|
|_______|
```

用`include`模板来做,是把头尾拿出来`header`,`footer`拿出来,
然后`@include('header')`,`@include('footer')`
需要`@include`两次,而继承则是把`header/footer`公共框架写在父模板中,继承一次父模板.

模板继承的概念和面向对象的继承非常相似,看下例.

```
<!-- 父模板 parent.blase.php -->
<html>
<meta charset="utf-8">
<body>
<div style="background:gray;">
  @section('leftside')
   this is parent left
  @show
</div>
<div style="background:green;">
  @section('rightside')
   this is parent right
  @show
</div>
</body>
</html>
```

父模板中有2个方法`leftside`,`rightside`;
子模板继承父模板,并且重写`leftside`,`rightside`方法,则可
获得子类的特定输出.

```
<!-- 子模板 son.blade.php-->
@extends('parent')
@section('leftside')
  son left
@endsection

@section('rightside')
  son right
  @parent
@endsection
```

根据面向对象的知识,子模板的同名方法覆盖父类方法.
同时,子类`rightside`方法中引用的父类方法.
因此,显示结果为:

```
<!-- 父模板 parent.blase.php -->
<html>
<meta charset="utf-8">
<body>
<div style="background:gray;">
  son left
</div>
<div style="background:green;">
  son right
  parent right
</div>
</body>
</html>
```

8.10 不解析标签

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}}
```

8.11 模板输出已自动防XSS攻击

`['code'=>'<script>alert(1)</script>']`
输出到view层
`<script>alert(1)</script>`

如果确实不需要实体转义,可以加`!!`.(1个大括号,不是两个)
例:`{!!$code!!}`

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`<project>/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`xxs`,复数形式.
如用户(user)表叫`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`

以`msgs`表对应的`Msg.php`文件为例,典型的Model如下:

```
namespace App;
use Illuminate\Database\Eloquent\Model;

class Msg extends Model
{
    //
}
```

9.3 自动生成Model

Model可以手写,可以也用artisan命令行工具生成.
例: `php artisan make:model Msg`
`php artisan make:model Msg -m` 生成Model同时执行migrate

9.4 实例化Model

```
$model = new App\Xxx(); // 得到Xx表的Model,且不与表中任何行对应.
```

```
$model = Xxx::find($id); // 得到Xx表的Model,且与$id行数据对应.
```

9.5 增

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

9.6 查

查单行: find()与first()

```
// 按id查
Msg::find($id) // 按id查

// 按where条件查
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
Msg::where('id','>',2)->get(['列1','列2']);
```

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

```
  public function del($id) {
    $msg = Msg::find($id);
    return $msg->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;
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');
```

分组(错)

```
Goods::groupBy('cat_id')->get(['cat_id','avg(price)']) );
//想要效果: select cat_id,avg(price) from goods group by cat_id;
//实际效果: select `cat_id`,`avg(price)` from goods group by cat_id;
//原因: laravel在字段名两边用反引号包住了.
```

分组(对)

```
// 用DB::raw()方法,raw是"裸,不修饰的"意思
Goods::groupBy('cat_id')->get(['cat_id',DB::raw('avg(price)')]) );
```

更复杂的查询表达式:
https://laravel.com/docs/5.1/queries

9.10 model的约定

- 表名的约定 默认表名为`Model`名+`s`,可能通过的model类的`table`属性来指定表名. 列:

```
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,你可以通过`primaryKey`属性来指定主键列名.

```
class XxModel extends Model {
    protected $primaryKey = 'Xx_id';
}
```

- `created_at`,`updated_at`字段 Model默认有这2个字段,且在更新行时,会自动帮你更新这两个字段. 如果不想这样,甚至不想要这2个字段,可以把model的`timestamps`属性设为`false`

```
class XxModel extends Model {
    public $timestamps = false;
}
```

10章 Request对象

`Request`对上放置着此次请求的全部信息.如:

- 请求方式(get/post)
- 请求参数($_POST,$_FILES)
- 请求路径 (域名后的部分)
- 请求cookie 等诸多信息,都存到的`Request`对象上

10.1 声明Request对象

在方法中,声明第1个参数为`Request`类型参数,即可自动接收.
Request作为方法的第1个参数出现.
另: 如果方法中有路由器绑定的参数,不受影响.
例:

```
Route::get('/del/{$id}');
public function del(Request $request , $id) {
    // $id参数虽然到第2个参数去了, 但不会受影响.
}
```

10.2 利用Request对象修改留言

例,用`Request`对象改进留言修改功能:

```
use Illuminate\Http\Request;
...
...
 public function up(Request $request , $id) {
    if( empty($_POST) ) {
      $msg = Msg::find($id);
      return view('msg.up',['msg'=>$msg]);
    }else {
      print_r( $request->all() );
      $msg = Msg::find($id);
      $msg->title = $request->input('title'); // input(POST参数)
      $msg->content= $request->content; // 直接访问属性
      $msg->pubtime = $request->input('pubtime',time());// 给个默认值

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

10.3 利用Request对象上传

```
// template <project>/resource/views/msg/add.php
<html>
<meta charset="utf-8">
<body>
<h1>laravel添加留言</h1>
```

```
<form action="<?php echo url('msg/add');?>" method="post">
  <p><input type="text" name="title"></p>
  <p>
    <textarea name="content"></textarea>
  </p>
  <p><input type="file" name="pic"></p>
  <p><input type="submit" value="提交"></p>
</form>
</body>
</html>
```

发布页:

```
public function post(Request $request) {
  $request->file('pic')->move('/path/to/image/xx.jpg');;
}
```

10.4 laravel与TP对比

- 路由器的区别 laravel的路由简单,灵活,指向控制器的方法. 而TP的路由是由`模块/控制器/方法`这种规律生成. 准确的说,TP不能叫路由,只是`URL`与控制器的对应关系

而TP的"规则路由","正则路由",只是`URL`的一个别名甚至是跳转链接,不是真正的路由.
而且路由只能在模块下发挥作用,容易出错.

```
'URL_ROUTER_ON'   => true,
'URL_ROUTE_RULES'=>array(
    'goods/:goods_id'=> 'Index/goods',
)
```

假如上例写在Home模块下,是指:
`Home/goods/3`--> `Home/Index/goods/goods_id/3`

- 整体设计的区别 laravel接管了网站的全过程,数据库+MVC+错误处理.

```
|----laravel------|
| DB+MVC+error    |
|-----------------|
```

而tp,不包含数据库管理

```
DB,table,columns不在tp管理范围
|----thinkphp-----|
|    MVC+error    |
|-----------------|
```

laravel更像一个全自动车床,输入原料,得到成品.
tp则部分需要手动,更像一个工具箱.

- 设计思想的区别 laravel"大处省流程",tp"小处省字母" 例: 而tp则`$_GET`,`$_POST`,仍是`$_GET`,`$_POST`, 仍需要手动接收,`I('get.id')`相比`$_GET`没有本质变化.

TP下,和纯手写博客时的思路,没有根本变化,仍是接`$_GET`,`$_POST`,`$_FILES`
然后自行去判断处理,只是改变了写参数的方式.
而laravel,则是接收参数的方式都已经截然不同.
`GET---> XxController->method($id);`

TP提供的`D()`,`M()`,`I()`,等有改变你的工作方式,只是让略省几个字母.
laravel则从流程和方式上,改变和简化工作.

- 面向对象的区别 通过文件上传体现的比较明显, laravel是把"WEB功能封装在对象里", TP则是"封装对象帮你做web功能"

- 模板的区别 laravel 的模板语法比TP语法简单

11章 helper函数

laravel提供了系列好用的函数,大致分为以下几类.
https://laravel.com/docs/5.1/helpers

- 数组函数
- 字符串函数
- 路径函数
- URL函数
- 杂项函数

各举几例:

- `array_collapse()` 把多维数组拆散组成一维数组:

```
$array = array_collapse([[1, 2, 3], [4, 5, 6], [7, 8, 9]]);
// [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- `str_limit()` 取字符前几位,并添加省略号

```
$value = str_limit('The PHP framework for web artisans.', 7);
// The PHP...
```

- str_random() 生成随机字符串

```
$string = str_random(8); // 8位随机字符串
```

- `e()` 实体转义

```
echo e('<html>foo</html>');
// &lt;html&gt;foo&lt;/html&gt;
```

- `app_path()` 返回当前项目的`app`目录的绝对路径
  `$path = app_path();`
  也可以用于获取`app`目录下的其他文件的路径.
  `$path = app_path('Http/Controllers/Controller.php');`

- `base_path()` 返回项目的绝对路径
  `$path = base_path();`
  也可以返回项目目录下某文件的绝对路径,例
  `$path = base_path('vendor/bin');`

- `config_path()` 返回项目的配置文件所在目录
  `$path = config_path();`

- `public_path()` 返回项目的公共文件所在目录(js,css等一般放这儿)
  `$path = public_path();`

- `url()` 生成规范URL

```
url('foo/bar'); // xx.com/foo/bar
url('/',[cat,3,'page',4]); // xx.com/cat/3/page/4
```

- `action()` 配合路由器,生成规范URL

```
//如果路由器没定义到XxController@method的路径,则会报错
action('XxController@method');
echo action('MsgController@del',[3,'page'=>4]);
```

- `back()` 退回上一页

```
return back();
```

- `bcrypt()` 加密密码

```
$password = bcrypt('my-secret-password');
```

- `config()` 读取配置值

```
$value = config('app.timezone');
$value = config('app.timezone', $default); //没读到配置,则返回$default
```

- `csrf_token()` 生成防跨域提交的随机串

```
$token = csrf_token();
```

- `csrf_field()` 生成防跨域提交的隐藏字段

```
{!! csrf_field() !!}
```

- `dd()` 打印变量并终止执行,一般调试时用

```
dd($value);
```

- `redirect()` 重定向

```
return redirect('/home');
```

- `request()` 得到当前的request对象

```
$request = request();
$value = request('key', $default = null)
```

12章 p2p网贷项目开发

12.1 功能分析

p2p , e租宝, 人人贷.

商业模式:
在豪华地段,租最豪华办公室,招模特做前台,一水的170,大长腿.
注册资本,怎么着也得一个亿吧,反正不用真实出资.

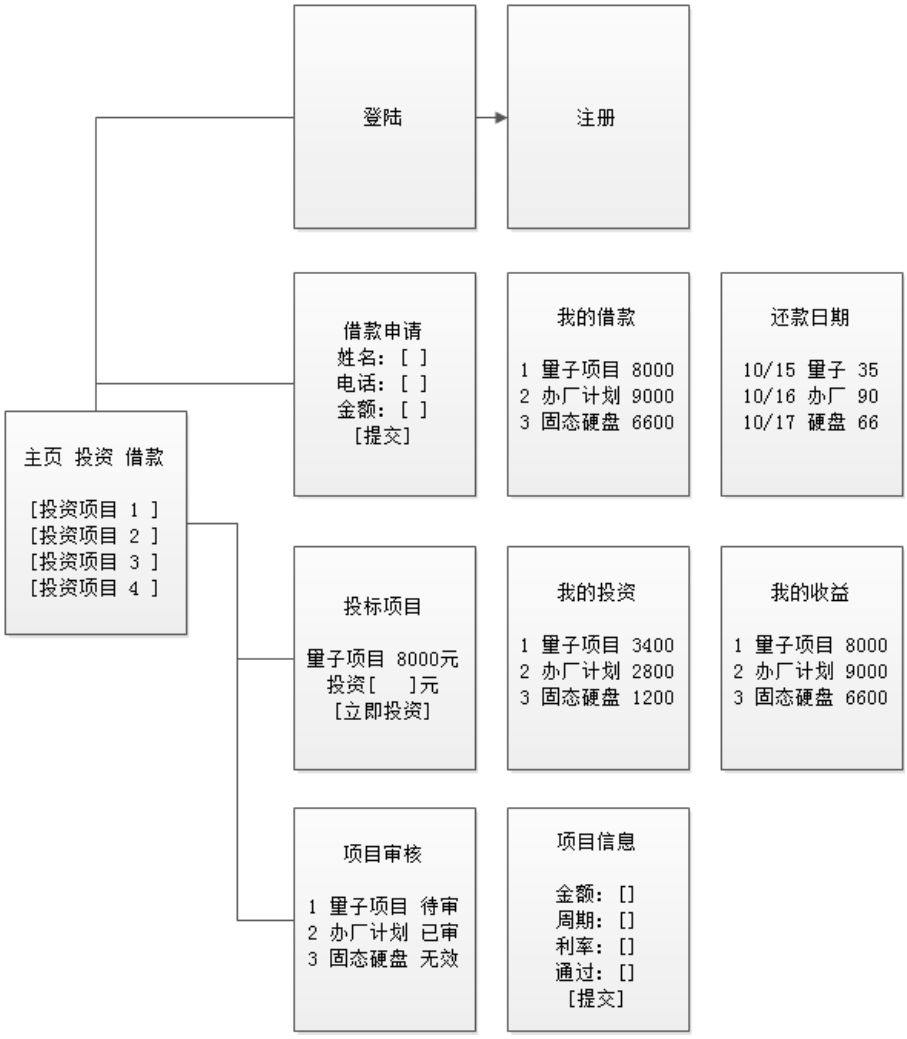
再到敬老院雇一重病老头做法人代表(关键!).

市场宣传,承诺高额回报,利率15%起,我说的是月利息!
没客户怎么办? 在央视黄金时段砸广告. 线下销售,电话销售,狂拉单子.
付不出利息怎么办? 没事,用下一个客户的本钱付上个客户的利息.

当客户本金积累到10亿时,法人代表及时的死掉.

我们的钱哪去了? 鬼知道!

基本功能:



12.2 准备模板

把点点贷模板解压放在`<project>/resources/view`下,
把解压出的`image`,`css`目录移动到`<project>/public/`目录下

写一个简单路由器测试:
`Route::get('test/index' , function(){return view('/index');});`
并把`index.html` 重命名为 `index.blade.php`

发现页面可以输出,但变形了,这是因为css文件路径不对.
浏览器`f12`打开控制台,以`common.css`为例:
`xx.com/test/common.css 404`

这是因为当前URL是`xx.com/test/index`, 而源码中`<link type="text/css" rel="stylesheet" href="css/common.css">`

所以形成了对`xx.com/test/css/common.css`的请求.

我们把所有模板的css,image路径,都换成相对于网站的根目录`/`就可以避免此问题.

批量替换所有模板`css/` --> `/css/`, `image/`-->`/image/`
再次刷新, 页面已正常显示.

12.3 表分析

用户表: users

字段	类型	说明
uid	primary key	主键
name	string	用户名
email	string	电子邮箱

mobile	string(11)	手机号
password	string(60)	密码
regtime	integer	注册时间
lastlogin	integer	上次登陆时间
rember_token	rememberToken()	记录用户cookie

借款表:projects 用到时再分析
借款附属表:atts
投资表:bids
收益表:grows
还款表:hks
流水表:logs

12.3 迁移文件

```
// users表
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->increments('uid');
        $table->string('name');
        $table->string('email')->unique();
        $table->string('mobile',11);
        $table->string('password', 60);
        $table->rememberToken();
        $table->integer('regtime');
        $table->integer('lastlogin');
    });
}
```

13章 认证与授权

13.1 准备工作

laravel自带了用户认证与授权类,可以方便的帮我们完成认证与授权.
主要用到:`App\Http\Controllers\Auth\AuthController`以及`users`表

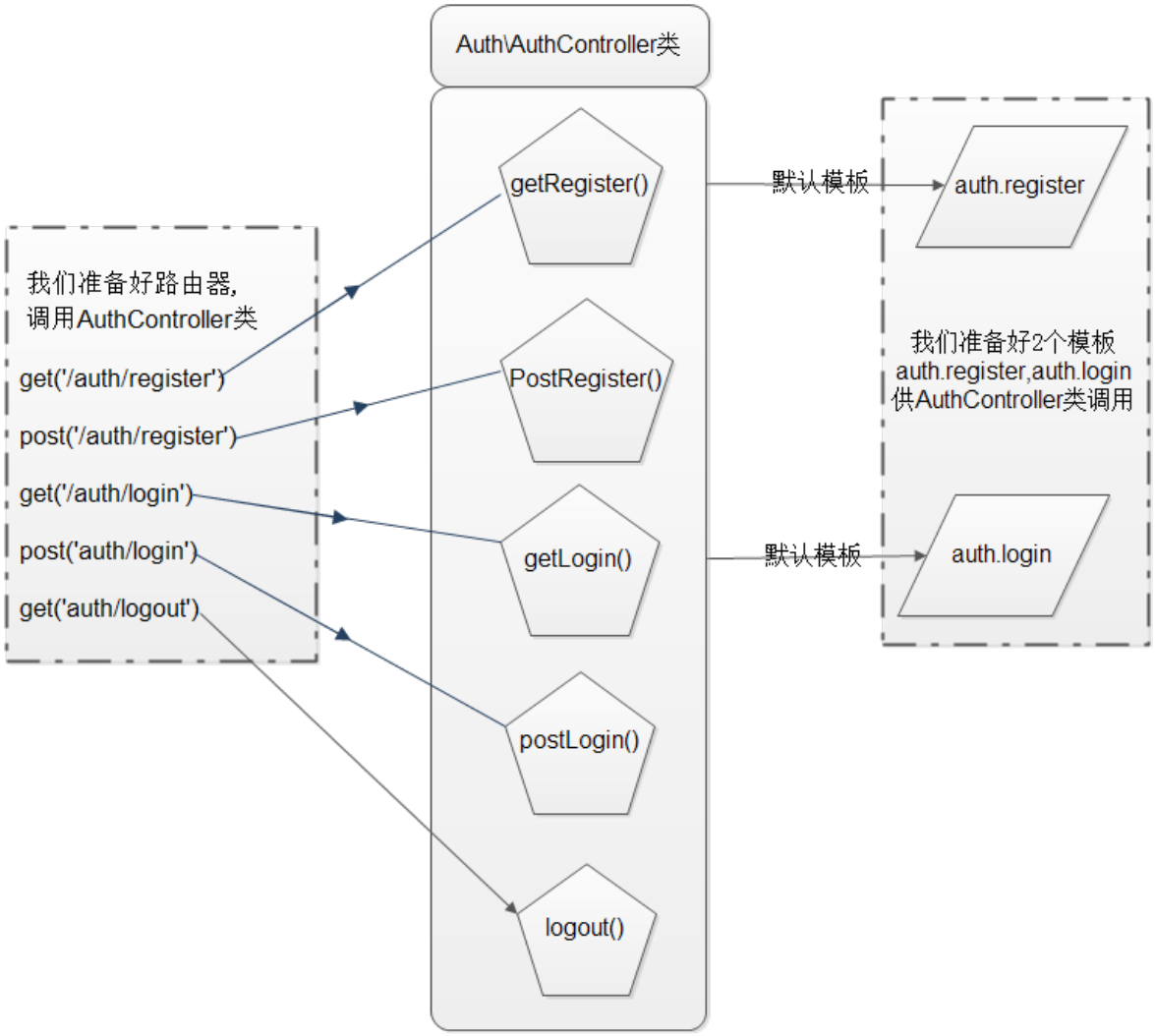
laravel要求`users`表至少有如下字段:

- name字段
- email字段
- password 60个长度
- remember_token 100个长度

由于我们的`uesrs`表主键叫`uid`,且没有`created_at`,`update_at`字段,
因此,按Model的约定,做如下说明.

```
protected $primaryKey = 'uid';
public $timestamps = false;
```

13.2 AuthController类的工作原理



`AuthController`类有几个重要属性,意义如下(如果需要,你可以修改这几个属性值).

```
//  依据版本5.1.33
class AuthController {
    protected $username = 'email'; #与password字段配合登陆的字段,name,email,mobile都可以
    protected $redirectPath = '/home'; # 登陆成功后的跳转方向
    protected $redirectAfterLogout = '/'; # 默认退出后跳转页
    protected $loginPath = 'auth/login'; #默认登陆URL
}
```

13.3 准备路由与模板:

```
// Registration routes...
Route::get('auth/register', 'Auth\AuthController@getRegister');
Route::post('auth/register', 'Auth\AuthController@postRegister');

// Authentication routes...
Route::get('auth/login', 'Auth\AuthController@getLogin');
Route::post('auth/login', 'Auth\AuthController@postLogin');
Route::get('auth/logout', 'Auth\AuthController@getLogout');
```

在`<project>/resources/view`下建立`auth`目录,
把`register.html`和`login.html`放入`auth`目录下,
并重命名为`register.blade.php`和`login.blade.php`.

13.4 用户注册

```
<form method="POST" action="{{url('auth/register')}}">
    {!! csrf_field() !!}
    <div>
       用户名:<input type="text" name="name" value="{{ old('name') }}">
    </div>
```

```
    <div>
       Email:<input type="email" name="email" value="{{ old('email') }}">
    </div>
    <div>
        手机:<input type="text" name="mobile">
    </div>
    <div>
        密码:<input type="password" name="password">
    </div>
    <div>
        确认密码:<input type="password" name="password_confirmation">
    </div>
    <!--注意,确认密码必须叫password_confirmation,才能自动验证-->
    <div>
        <button type="submit">注册</button>
    </div>
</form>
```

13.5 手机号丢失了

注册用户的手机号为空,如下:

```
+-----+-------+--------------+-------------+
| uid | name  | email        | mobile      |
+-----+-------+--------------+-------------+
|   1 | test1 | test1@qq.com |             |
|   2 | test2 | test2@qq.com |             |
|   3 | test3 | test3@qq.com |             |
+-----+-------+--------------+-------------+
```

追踪`Auth\AuthController`类的源码:

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
    ]);
}
```

发现在创建用户时,没有传递`mobile`字段.
修改如下:

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'mobile' => $data['mobile'],
        'password' => bcrypt($data['password']),
    ]);
}
```

再次提交,仍然收不到`mobile`字段,再次查看Model `User.php`的文件,有如下属性:
`protected $fillable = ['name', 'email' , 'password'];`
这个属性代表,外界对`User Model`批量赋值时,`Model`接收的字段.
修改如下:
`protected $fillable = ['name', 'email' , 'mobile', 'password'];`

再次注册,此时数据库有`mobile`数据了.

```
+-----+-------+--------------+-------------+
| uid | name  | email        | mobile      |
+-----+-------+--------------+-------------+
|   4 | test4 | test4@qq.com | 12222222222 |
+-----+-------+--------------+-------------+
```

13.6 用户登陆

```
<form method="POST" action="{{url('auth/login')}}">
    {!! csrf_field() !!}
    <div>
       用户名:<input type="text" name="name" value="{{ old('name') }}">
    </div
    <div>
        密码:<input type="password" name="password">
    </div>
    <div>
        <input type="checkbox" name="remember"> 记住我
    </div>
    <div>
        <button type="submit">登陆</button>
    </div>
</form>
```

13.7 登陆失败的处理

在laravel中,如果表单校验失败,会自动传递一个`$errors`变量到`view`中.
因此,我们可以在登陆模板中直接打印`$errors`来查看错误

在`<project>/resources/views/auth/login.blade.php`中添加一句
`{{print_r($errors)}}`,并再次登陆,会看到错误信息:

```
 Array ( [email] => Array ( [0] => The email field is required. )
```

这是因为: laravel框架默认用`email`+`password`字段来检测用户并登陆.
回头看13.2章节,修改`Auth\AuthController`的属性如下:
`protected $username = 'name';`
即,指定laravel以'用户名'和'密码'来登陆,即可.

13.8 手动登陆

如果有自己独特的独特登陆逻辑,可以手动登陆.

- 验证密码 attemp()

```
Auth::attempt(['email' => $email, 'password' => $password])
Auth::attempt(['name' => $name, 'password' => $password, 'active' => 1])
```

- 检测登陆状态
  `Auth::check()`

- 退出

```
Auth::logout();
```

14章 借款功能

14.1 借款表及迁移文件

借款表:projects

字段	类型	说明
pid	primary key	主键
uid	integer	用户uid
name	string	用户名
money	integer	贷款金额
mobile	string(11)	手机

title	string	项目名称
rate	tinyint	利率(百分比)
hrange	tinyint	还款期限,月为单位
status	tinyint	0审核中,1招标中,2还款中,3结束
revice	integer	已招标金额
pubtime	integer	项目发布时间

借款附属信息表:atts

字段	类型	说明
aid	primary key	主键
uid	integer	用户uid
pid	integer	项目pid
title	string	项目名称
realname	string	真实姓名
age	tinyint	年龄
gender	enum('男','女')	性别
salary	tinyint	千为单位
jobcity	string	工作城市
udesc	string	用户描述

```
// projects表 迁移文件
public function up()
{
    Schema::create('projects', function (Blueprint $table) {
        $table->increments('pid');
        $table->integer('uid');
        $table->string('name',10);
        $table->integer('money');
        $table->string('mobile',11);
        $table->string('title',50);
        $table->tinyint('rate');
        $table->tinyinteger('hrange');
        $table->tinyinteger('status'); # 0审核中,1招标中,2还款中,3结束
        $table->integer('recive');
        $table->integer('pubtime');

    });
}


// atts表 迁移文件
public function up()
{
    Schema::create('atts', function (Blueprint $table) {
        $table->increments('aid');
        $table->integer('uid');
        $table->integer('pid');
        $table->string('realname');
        $table->tinyinteger('age');
        $table->enum('gender',['男','女']);
        $table->tinyinteger('salary');
        $table->string('jobcity');
        $table->string('udesc');
```

```
    });
}
```

14.2 准备路由器和Controller

```
Route::get('jie' , 'ProController@jie');
Route::post('jie' , 'ProController@jiePost');
```

```
public function jie() {
  return view('woyaojiekuan');
}

public function jiePost() {
  return 'insert into DB';
}
```

14.3 生成Model

贷款申请牵涉到2张表,`projects`,`atts`;
我们为期生成Model.

```
php artisan make:model Pro
php artisan make:model Att
```

```
class Pro extends Model
{
    protected $table = 'projects';
    protected $primaryKey = 'pid';
    public $timestamps = false;

    protected $fillable = ['money','age','mobile'];
}

class Att extends Model
{
    protected $primaryKey = 'aid';
    public $timestamps = false;
    //
}
```

14.4 借款方法

```
public function jie(Request $request) {
 return view('woyaojiekuan');
}

public function jiePost(Request $request) {
 //$rs = Pro::create($request->all());

 $pro = new Pro();
 $pro->age = $request->age;
 $pro->money= $request->money;
 $pro->mobile= $request->mobile;
 $pro->pubtime = time();

 $rs = $pro->save();

 // 开始写入atts附属表信息
 $att = new Att();
 $att->uid = $user->uid;
 $att->pid = $pro->pid;
 $att->save();

 var_dump($rs);
}
```

14.5 获取登陆用户信息

在上一节中,发布的`Project`信息没有用户的`uid`,`name`信息.
我们需要从登陆信息中拿到这两条信息,并写入`Project`中.

laravel中,获取当前登陆用户的对象,可以用如下方法:

```
$user = Auth::user();
$user = $request->user();
```

通过`$user`实例,可以拿到当前登陆用户的信息.
因此,

```
public function jiePost(Request $request) {

 //$user = Auth::user();
 $user = $request->user();

 ...
 $pro->uid = $user->uid;
 $pro->name= $user->name;
 ...
}
```

15章　项目审核

15.1 审核列表页

路由器:

```
Route::get('prolist','CheckController@prolist');
```

控制器:

```
// file /app/Http/Controllers/CheckController.php
public function prolist() {
    $pros = Pro::orderBy('pid','desc')->get();
    return view('prolist' , ['pros'=>$pros]);
}
```

view

```
@forelse ($pros as $p)
<tr>
   <td class="f">{{$p->pid}}</td>
   <td>{{date( 'Y/m/d H:i' , $p->pubtime)}}</td>
   <td class="tr"><span>{{$p->name}}</span></td>
   <td class="tr">{{$p->money/100}}</td>
   <td class="tr">{{$p->mobile}}</td>
   <td class="">
    @if ($p->status == 0)
    待审核
    @elseif ($p->status == 1)
    招标中
    @elseif ($p->status == 2)
    还款中
    @elseif ($p->status == 3)
    已结束
    @endif
   </td>
   <td class="l"><a href="{{ url('check' , [$p->pid]) }}">审核</a></td>
</tr>
@empty
<div class="wujilu" id="errorMsg">暂无记录</div>
@endforelse
```

15.2 审核表单

路由器:

```
Route::get('check/{pid}' , 'CheckController@check');
```

控制器:

```
// 审核项目,主要是修改projects表和atts表
public function check($pid) {
    $pro = Pro::find($pid);
    $att = Att::where('pid' , $pid)->first();

    if(empty($pro)) {
        return redirect('/prolist');
    }

    return view('shenhe' , ['pro'=>$pro , 'att'=>$att]);
}
```

15.3 审核过程

路由器:

```
Route::post('check/{pid}' , 'CheckController@checkPost');
```

控制器

```
// 审核数据update到数据库
public function checkPost(Request $req , $pid){
    $pro = Pro::find($pid);
    $att = Att::where('pid' , $pid)->first();

    if(empty($pro)) {
        return redirect('/prolist');
    }

    $pro->title = $req->title;
    $pro->hrange = $req->hrange;
    $pro->rate = $req->rate; // 百分比
    $pro->status = $req->status;

    $att->realname = $req->realname;
    $att->gender = $req->gender;
    $att->udesc = $req->udesc;


    if( $pro->save() && $att->save() ) {
        return redirect('/prolist');
    } else {
        return 'error';
    }

}
```

16章 投资功能

16.1 投资表及迁移文件

投标表:bids

字段	类型	说明
bid	primary key	主键
uid	integer	用户uid
pid	integer	项目pid

title	string	项目名称
money	integer	投标金额
pubtime	integer	投标时间

```
// bids表 迁移文件
public function up()
{
    Schema::create('bids', function (Blueprint $table) {
        $table->increments('bid');
        $table->integer('uid');
        $table->integer('pid');
        $table->string('title',50);
        $table->integer('money');
        $table->integer('pubtime');
    });
}
```

16.2 在线投标

```
 public function touziPost(Request $request , $pid) {
     $pro = Pro::find($pid);
     $bid = new Bid();
     $user = Auth::user();

     $bid->uid = $user->uid;
     $bid->pid = $pro->pid;
     $bid->title = $pro->title;
     $bid->money = $request->input('v_amount');
     $bid->pubtime = time();

     $bid->save(); // 写入投资信息

     // 修改项目收到的金额
     $pro->increment('recive' , $bid->money);

     // 如果投标已满,则改此项目的状态.
     if($pro->money == $pro->recive) {
        //$pro->status = 2;
        //$pro->save()
        $this->touziDone($pro); //交给专门的方法来处理
     }

     echo 'succ';
  }
```

16.3 还款表与收益表

投标完成后, 借款人每月要还"本+息",投资人每天要收利息.
因此,我们再建2张表.

还款表:hks

字段	类型	说明
hid	primary key	主键
uid	integer	用户uid
pid	integer	项目pid
title	string	项目名称
amount	integer	每月还款金额
paydate	date	账单日
status	tinyint	是否已还

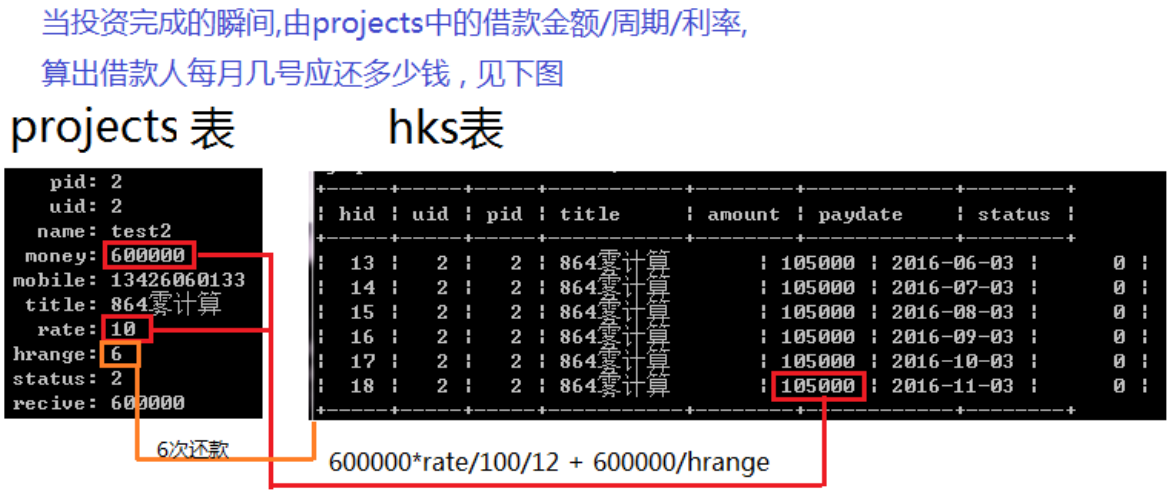
预收益表:tasks

字段	类型	说明
tid	primary key	主键
uid	integer	用户uid
pid	integer	项目pid
title	string	项目名称
amount	integer	每天的利息
enddate	date	收利息截止日

```
// 还款账单表 迁移文件
public function up()
{
    Schema::create('hks', function (Blueprint $table) {
        $table->increments('hid');
        $table->integer('uid');
        $table->integer('pid');
        $table->string('title');
        $table->integer('amount');
        $table->date('paydate');
    });
}
```

```
// tasks表迁移文件
public function up()
{
    Schema::create('tasks', function (Blueprint $table) {
        $table->increments('tid');
        $table->integer('uid');
        $table->integer('pid');
        $table->string('title');
        $table->integer('amount');
        $table->date('enddate');
    });
}
```

16.4 生成月账单



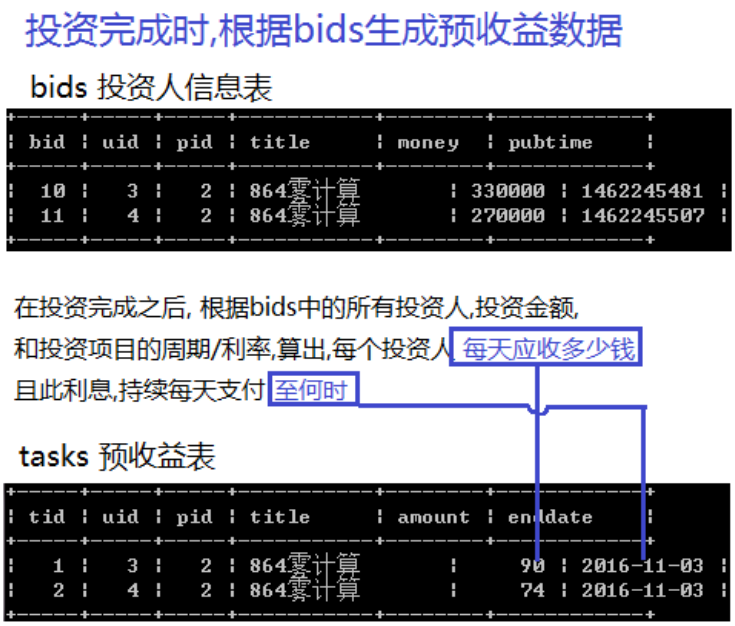

```
protected function touziDone($pro) {
    // 先修改此招标项目的状态
    $pro->status = 2;
    $pro->save();
```

```
    // 为借款人,n个月还款周期,生成N条还款记录.
    $amount =  ($pro->money * $pro->rate / 1200 )  + ($pro->money / $pro->hrange); // 算出每月还款额

    $row = ['uid'=>$pro->uid , 'pid'=>$pro->pid , 'title'=>$pro->title];
    $row['amount'] = $amount;
    $row['status'] = 0;

    $today = date('Y-m-d');
    for($i=1; $i<=$pro->hrange; $i+=1) {
        $paydate = date('Y-m-d' , strtotime("+ $i months"));
        $row['paydate'] = $paydate;
        DB::table('hks')->insert($row);
    }
}
```

16.5 生成预期收益




```
protected function touziDone($pro) {
    ....
    ....
    // 3. 为投资者,生成收益打款任务
    $bids = Bid::where('pid' , $pid)->get();
    $row = [];
    $row['pid'] = $pid;
    $row['title'] = $pro->title;
    $row['enddate'] = date('Y-m-d' , strtotime("+ {$pro->hrange} months"));

    foreach($bids as $bid) {
        $row['uid'] = $bid->uid;
        $row['amount'] = $bid->money * $pro->rate / 36500;
        DB::table('tasks')->insert($row);
    }
}
```

16.6 生成收益

每天收益表:grows

字段	类型	说明
gid	primary key	主键
uid	integer	用户uid
pid	integer	项目pid

title	string	项目名称
amount	integer	每天的利息
paytime	date	收益日期

```
// grows表迁移文件
Schema::create('grows', function (Blueprint $table) {
    $table->increments('gid');
    $table->integer('uid');
    $table->integer('pid');
    $table->string('title',60);
    $table->integer('amount');
    $table->date('paytime');
});
```

```
// GrowController.php
public function run() {
    $today = date('Y-m-d');
    $tasks = DB::table('tasks')->where('enddate' , '>=' , $today)->get();

    foreach($tasks as $t) {
        $t->paytime = $today;
        $t = (array)$t;
        unset($t['tid']);
        unset($t['enddate']);

        DB::table('grows')->insert($t);
    }
}
```

16.7 我的账单

```
public function myzd() {
    $hks = DB::table('hks')->get();
    return view('myzd' , ['hks'=>$hks]);
}
```

```
@foreach($hks as $h)
<tr>
   <td class="f">{{$h->hid}}</td>
   <td class="tr"><span>{{$h->title}}</span></td>
   <td class="tr">{{$h->amount/100}}</td>
   <td class="tr">{{$h->paydate}}</td>
   <td class="tr">
    @if ($h->status == 0)
    <a href="#">立即还款</a>
    @else
    已还
    @endif
    </td>
</tr>
@endforeach
```

16.8 我的投资

```
public function mytz() {
    $user = Auth::user();
    $bids = Bid::where('bids.uid' , $user->uid)->whereIn('status' , [1,2])
     ->join('projects' , 'bids.pid' , '=' , 'projects.pid')->get(['bids.*' , 'projects.status']);
    //dd($bids);
    return view('mytz' , ['bids'=>$bids]);
}
```

```
@foreach($bids as $b)
<tr>
   <td class="f">{{$b->pid}}</td>
```

```
    <td class="tr"><span>{{$b->title}}</span></td>
    <td class="tr">{{$b->money/100}}</td>
    <td class="tr">{{date('Y/m/d' , $b->pubtime)}}</td>
    <td class="tr">
     @if ($b->status == 1)
     招标中,快叫你的朋友来投资
     @elseif ($b->status == 2)
     收益中,发了
     @elseif ($b->status == 3)
     已结束
     @endif
    </td>
</tr>
@endforeach
```

16.9 我的收益

```
public function mysy() {
    $user = Auth::user();
    $grows = DB::table('grows')->where('uid' , $user->uid)->orderBy('gid' , 'desc')->get();

    return view('mysy' , ['grows'=>$grows]);
}
```

```
@foreach($grows as $g)
<tr>
    <td class="f">{{$g->gid}}</td>
    <td class="tr"><span>{{$g->title}}</span></td>
    <td class="tr"> {{$g->amount/100}}元 </td>
    <td class="tr">{{$g->paytime}}</td>

</tr>
@endforeach
```

17章 中间件

中间件可以任意穿插在"请求-响应"过程中,更直接的说,可以插入到`XxController@Method()`方法前或后去执行.
以影响执行的结果,从而让你的代码更加灵活.

17.1 生成中间件

`php artisan make:middleware EmailMiddleware`
生成的中间件位于
`<project>/app/Http/Middleware/EmailMiddleware`

```
namespace App\Http\Middleware;
use Closure;

class EmailMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        echo 'I will sent an email to you!';
        return $next($request);
    }
}
```

17.2 注册中间件

```
// 网上(包括官网)的例子全是错的!
```

```
Route::get('home' ,['uses'=>'MsgController@home',
'middleware'=>['App\Http\Middleware\EmailMiddleware']] );
```

源码在下面:

```
// file: laravel/framework/src/Illuminate/Routing/Router.php
protected function actionReferencesController($action)
{
    if ($action instanceof Closure) {
        return false;
    }

    return is_string($action) || is_string(isset($action['uses']) ? $action['uses'] : null);
}
```

17.3 用中间件检测登陆

中间件可以注册在单个路由上:

```
Route::get('/', ['middleware'=>['App\Http\Middleware\EmailMiddleware'],'uses'=>'IndexController@index']);
```

中间件也可以注册在全局上:

```
// 全局路由是在<project>/App/Http/kernel.php上指定
/**
 * The application's global HTTP middleware stack.
 *
 * @var array
 */

// 例:这是系统注册的几个全局路由
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \App\Http\Middleware\EncryptCookies::class,
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    \App\Http\Middleware\VerifyCsrfToken::class,
];
```

中间件也可以注册在控制器上

```
XxController  {
  protected $middleware = ['App\Http\Middleware\Authenticate'=>[]];
}
```

17.4 注册后发邮件

安装email类: `composer require nette/mail`

声明路由器:

```
Route::post('auth/register',
[
'middleware'=>'App\Http\Middleware\EmailMiddleware',
'uses'=>'Auth\AuthController@postRegister'
]
);
```

```
// Http/Middleware/EmailMiddlewar
use Nette\Mail\Message;
use Nette\Mail\SmtpMailer;

public function handle($request, Closure $next)
{
    $rs = $next($request);

    // 注册后发邮件
```

```
    //$request->user()->email;

    if($request->user()) {

        $mail = new Message;
        #注意,下行的setFrom要和你的邮箱名保持一致
        $mail->setFrom('laowang <superkeysir@163.com>')
            ->addTo($request->user()->email)
            ->setSubject('试试我的中间件')
            ->setBody("真好用!");

        $mailer = new SmtpMailer(array(
        'host' => 'smtp.163.com',
        'username' => 'superkeysir',#你的163用户名
        'password' => 'admins'# 你的邮箱密码
        ));
        $mailer->send($mail);

    }

    return $rs;
}
```

17.5 指定某个路由不用csrf检测

https://mattstauffer.co/blog/excluding-routes-from-the-csrf-middleware-in-laravel-5.1

18章 在线支付

详见"在线支付笔记"

19章 功能细化

19.1 登陆状态

在view层,判断用户是否登陆

```
 @if(!Auth::user())
<a class="fff" title="登录" rel="nofollow" href="{{url('auth/login')}}">登录</a>
<a class="fff" title="注册" rel="nofollow" href="{{url('auth/register')}}">注册</a></div>
@else


<em class="fff fs_12">您好，</em><a href="#" rel="nofollow" class="hello">{{Auth::user()->name}}</a>
<a href="{{url('auth/logout')}}" id="loginOut" class="hello">退出</a></div>
<dl>
    <dt><a class="txnone" title="账户中心" rel="nofollow" target="_blank" href="/home">账户中心</a></dt>
</dl>
@endif
```

19.2 分页功能

以'我的账单'功能为例,不分页是这样查询的`DB::table('hks')->get()`
如果需要分页,每页2条,则`DB::table('hks')->paginate(2)`

此时,laravel会按当前页取出合适的数据,同时,生成分页链接.
分页链接用`render()`方法获取.

实例:

```
public function myzd() {
    $hks = DB::table('hks')->paginate(2);
    return view('myzd' , ['hks'=>$hks]);
}
```

```
{{$hks->render()}}
```

19.3 JS验证

以发布借款项目为例,做JS验证

```
$('#applyForm').submit(function(){
   var patt;

   if( $('#age').val() == '0' ) {
    alert('请选择年龄');
    return false;
   }

    patt = /^[1-9]\d*$/
    if( !patt.test( $('#loanAmount').val() ) ) {
        alert('金额必须是整数');
        return false;
    }

    patt = /^1[3578]\d{9}$/
    if(!patt.test( $('#mobile').val() )) {
        alert('手机号不正确');
        return false;
    }

   //return false;
});
```

20 自动验证

20.1 验证案例

第1个参数为`Request`对象,第2个参数为**验证规则**
验证规则为关联数组,语法如下:

```
[
'待验证字段'=>'验证规则',
]
```

借款验证案例:

```
// file: <project>/app/Http/Controllers/ProControll.php
public function jiePost(Request $req) {
    $this->validate($req , [
        'age'=>'required|in:15,40,80',
        'money'=>'required|digits_between:2,7',
        'mobile'=>'required|regex:/^1[3458]\d{9}$/',
        ]
        );
}
```

验证未通过的检测,以`money`为例

```
<span>借款金额</span>
<input id="loanAmount" name="money" maxlength="8" placeholder="请输入借款金额" type="text" value="{{old('money')}}">
@if ($errors->has('money'))
<p style="display: block;" id="amountError" class="error">{{$errors->first('money')}}</p>
@endif
```

20.2 验证规则

常用验证规则见下:
https://www.laravel.com/docs/5.1/validation#available-validation-rules

laravel的验证规则可以写多个,用'|'隔开.
例:`'money'=>'required|digits_between:2,7'`
是指: `money`字段必须存在,且位数在`2,7`之间.

20.3 自定义错误信息

如果验证未通过,需要自定义错误信息,只需在第3个参数中传递.

```
public function jiePost(Request $req) {
    /*
    $this->validate($req , [
        'age'=>'required|in:15,40,80',
        'money'=>'required|digits_between:2,7',
        'mobile'=>'required|regex:/^1[3458]\d{9}$/',
        ] ,

        [
        'money.required'=>'money必须写',
        'money.digits_between'=>'填的啥玩意儿?',
        'in'=>':attribute 必须是 :values 之一',
        'regex'=>':attribute 不合要求'
        ]

        );
}
```

20.4 手动验证

如果不用`$this->validate()`方法,也可以手动来创建一个`验证对象`

```
// 用Validator::make(数据,规则,错误提示);
$validate = Validator::make($req->all() ,

     [
    'age'=>'required|in:15,40,80',
    'money'=>'required|digits_between:2,7',
    'mobile'=>'required|regex:/^1[3458]\d{9}$/',
    ],

    [
    'money.required'=>'money必须写',
    'money.digits_between'=>'填的啥玩意儿?',
    'in'=>':attribute 必须是 :values 之一',
    'regex'=>':attribute 不合要求'
    ]
    );

// ->fails()负责判断,是否失败
if( $validate->fails() ) {
    // 回退到上一页,且携带出错数据,和上次表单中的旧数据
    return back()->withErrors($validate)->withInput();
}
```

19.5 短信验证码

```
// 短信发送方法
public function sm(Request $request ,  $mobile) {
    // 注意,把阿里短信SDK放在/vender下
    require (base_path(). '/vendor/alidayu/TopSdk.php');

    $appkey = '你的阿里大鱼appkey';
    $secret = '你的阿里大鱼secret密钥';

    $c = new \TopClient; // 前面的'/'代表根空间,不能少
    $c->appkey = $appkey;
    $c->secretKey = $secret;
    $rand = mt_rand(1000,9999); // 随机验证码

    // 把
    $request->session()->put('smscode' , $rand);

    $req = new \AlibabaAliqinFcSmsNumSendRequest;
    $req->setExtend("123456");
```

```
    $req->setSmsType("normal");
    $req->setSmsFreeSignName("布尔教育");
    $req->setRecNum($mobile);
    $req->setSmsParam("{\"code\":\"{$rand}\",\"product\":\"点点贷\"}");
    $req->setSmsTemplateCode("SMS_4715200");

    //print_r($req);
    echo $c->execute($req); // 输出结果,默认json格式
}
```

20章 artisan工具

20.1 生成artisan命令

如下:生成一个`grow`命令

`php artisan make:console --command=grow`

此时自动生成`<project>/app/Console/Commands/Grow.php`

20.2注册artisan命令

打开`<project>/app/Console/Kernel.php`

```
protected $commands = [
    Commands\Inspire::class,
];
```

修改为:

```
protected $commands = [
    Commands\Inspire::class,
    Commands\Grow::class,
];
```

20.3 完成命令

命令的内容写在`handle()`方法中,例:

```
public function handle() {
    // 你的业务逻辑....
    echo "growing";
}
```

20.4 带色彩的输出

在Linux的终端上,支持字符彩色输出.

例:

`$this->info("growing")`,则显示绿底文字

`$this->error("growing")`,则显示红底文字

20.5 定时任务

Linux: crontab

Win: 控制面板->定时任务

21章 路由分组(学员讲解)

laravel路由支持分组,如下例:

```
Route::group(['prefix'=>'auth' , 'namespace'=>'Auth'] , function(){

    Route::get('login' , 'AuthController@getLogin');
    Route::post('login' , 'AuthController@postLogin');
    Route::get('logout' , 'AuthController@getLogout');

});
```

等价于:

```
Route::get('auth/login' , 'Auth\AuthController@getLogin');
Route::post('auth/login' , 'Auth\AuthController@postLogin');
Route::get('auth/logout' , 'Auth\AuthController@getLogout');
```

21.2 项目技术点总结

环境:Wamp+sublime / Linux+vim
框架: laravel
技术点: composer,laravel,正则,权限控制,中间件,artisan,定时任务,Js,Ajax,在线支付,表单验证, laravel自动验证，邮件发送