第1章 如何安装laravel

1.1四种安装方法

我们该如何安装laravel,如果从网上搜索的话,你大概能够找到四种方法,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=5.1.33`

2 . 手动下载 laravle 本身 ,composer 安装依赖库 (半自动化)
https://github.com/laravel/laravel/tree/5.1
下载laravel,再到项目目录下,执行`composer install`;

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；
雷锋在这 : http://www.golaravel.com/download/
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 . larvel 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>
```

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>
```

修改hosts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>/app/Http/routes.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

```
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
    }
    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);
}
```

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.

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

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

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；

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

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：

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:

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 :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);
```

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7章 完整的增删改查--留言板

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');
```

生成控制器 ： php artisan make:controller MsgController
在控制器中，补全实现方法；

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');
}
```

3.执行迁移命令： php artisan migrate

7.3 发布留言

表单页 /resources/views/msg/add.php

```
<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>
```

```
public function add(){
    return view('msg.add');
}
```

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(); ?>">
```

添加方法：

```
public function addPost()
```

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

报错，是因为use DB；

7.4 显示留言列表

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

/resources/views/msg/index.php ：

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

7.5 删除留言

模板修改：

```
<td>
    <a href="/msg/del/<?php echo $m->id;?>"> 删除 </a>
    |
    <a href="/msg/up/<?php echo $m->id;?>"> 修改 </a>
</td>
```

删除+跳转

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

7.6 修改留言板

1.取出数据，在模板中显示

```
public function up($id)
```

```
{
    if(empty($_POST)){
        $row = DB::table('msgs')->where('id',$id)->first();
        return view('msg.up',['row'=>$row]);
    }
}
```

\/resources/views/msg/up.blade.php

```
<p>
    <input type="text" name="title" value="<?php echo $row->title; ?>">
</p>
<p>
    <textarea name="content"><?php echo $row->content; ?></textarea>
</p>
```

提交数据，写入数据库

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

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

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 :

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

forelse 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：
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；
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('left')
        this is parent left
        @show
    </div>
    <div style="background:green;">
        @section('right')
        this is parent right
        @show
    </div>
</body>
</html>
```

父模板中有 2 个方法left , right;
子模板继承父模板 , 并且重写left,right方法,
即可获得子类的特定输出.

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


@section('right')
    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right 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>
```

8.5 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xxs,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
```

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
{
    //
}
```

Controller中只需要 正常new 就可以了：

```
new \App\Msg()
```

9.3 自动生成和实例化

Model 可以手写,可以也用 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
}
```

- created_at,updated_at字段的约定
  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,
  可以在创建迁移文件后，删除 $table->timestamps();
  然后，设置 model 的timestamps 属性设为false

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

第10章laravel缓存的应用

什么是缓存,缓存的应用是什么,比如在php中有很多种缓存全页面静态化缓存,查询缓存,数据缓存等等;
那么我们简单的理解就是,比如说我查询一条数据,但是这个文件的内容很多,直接去查会浪费很多时间,那么我就可以去生成缓存,下次再去查询直接从缓存文件中去查,然而在larave框架中呢,也封装好了缓存类,直接使用即可;如下:
Laravel为不同的缓存系统提供了统一的API。缓存配置位于`config/cache.php`。在该文件中你可以指定在应用中默认使用哪个缓存驱动.laravel默认使用的缓存驱动是`file`,Laravel 支持当前流行的缓存后端，如 Memcached 和 Redis。我们学习的是它的默认缓存驱动`file`.

10.1缓存怎么样使用？

我们可以在控制器中去使用,只需要`use Illuminate/Support/Facades/Cache`就可以,
那用法是什么呢?静态方法调用方法即可
很简单`Cache::+方法` 如:存储缓存

```
Cache::put('key','value',$minutes);
```

10.2存储缓存

可以使用Cache门面中的put方法,当你在缓存中存储缓存项的时候，你需要指定数据被缓存的时间（分钟数）：
`Cache::put('key','value',$minutes);`

```
//三个参数
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;
use Illuminate/Support/Facades/Cache;

class TestController extends Controller
{
   public function cache1(){
    Cache::put('key1','val1',10);
   }

//add方法只会在缓存项不存在的情况下添加缓存项到缓存，如果缓存项被添加到缓存返回true，否则，返回false：
  public function add(){
    /*
    $ca = Cache::add('key1','val1',10);
    var_dump($ca);//因为之前存储过,所以返回false
    */
    $ca = Cache::add('key2','val2',20);
    var_dump($ca);//返回true
  }

//forever方法用于持久化存储缓存项到缓存，这些值必须通过forget方法手动从缓存中移除：
    Cache::forever('key','val');
```

以上呢是存储缓存,放在`storage/framework/`下

10.3 获取缓存

Cache门面的get方法用于从缓存中获取缓存项，如果缓存项不存在，返回null。如果需要的话你可以传递第二个参数到get方法指定缓存项不存在时返回的自定义默认值：

```
   public function huoqu(){
/*
    $v = Cache::get('key1');
    var_dump($v);//val2,上一步存储的缓存
*/
    $v = Cache::get('key1','default');
    var_dump($v);//如果有缓存,就输出缓存的值,否则输出defaul
}
```

还可以作回调使用，如果缓存项不存在的话可以自动调用去添加一个缓存。
如:

```
    $v = Cache::get('key1',function(){
    return Cache::put('key1','val1',10);
});
    var_dump($v);//返回val1
```

//如果你需要从缓存中获取缓存项然后删除，你可以使用pull方法，和get方法一样，如果缓存项不存在的话返回null：
如::

```
   Cache::pull('key1');
```

10.4删除缓存数据

可以使用`forget`方法

```
    $v = Cache::forget('key1');
    var_dump($v);//NULL
```

//也可以用`flush`方法清空所有缓存

```
    Cache::flush();
```

//可能会遇到这种情况,我们要删除的缓存数据没有
//可以通过`has`方法去判断

```
if(Cache::has('key1')){
    //如果有
    Cache::forget('key1');
}else{
    //没有
    echo 'error';
}
```

第11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

11.1 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.
}
```

11.2 利用Request对象修改留言

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);
        // 直接访问属性
        $msg->title = $request->title;
        $msg->content= $request->content; //
        return $msg->save() ? 'OK' : 'fail';
    }
}
```

也可以调用input 函数：
修改上一段代码

```
$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(好处就在此)
```

11.3 利用Request对象做文件上传

路由：

```
Route::get('msg/fil','MsgController@fil');
Route::post('msg/ups','MsgController@ups');
```

控制器：

```
public function fil(){
    return view('msg.fil');
}
public function ups(Request $req){
    // $req = request();
    $req->file('photo')->move('D:/xampp/htdocs/myphp/demo/dddai/public/','bb.png');
}
```

模板：

```
<form class="" action="/msg/ups" method="post" enctype="multipart/form-data">
    <input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
    <input type="file" name="photo" value="">
    <input type="submit" value="zou">
</form>
```

11.4 laravel 与 TP 对比

路由器的区别：
laravel 的路由简单,灵活,直接指向控制器的方法.
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，或者叫URL分发；
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.

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 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.

模板的区别 laravel 的模板语法比 TP 语法简单；