Laravel 5.5 入门教程

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

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);
```

在laravel中，return不能直接返回true或false

插入多行 (多维数组)

插入成功也是返回true，失败返回false。

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

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

insertGetId()方法只能添加一条数据，多条数据会出错。

```
$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；

返回受影响的行数

```
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());

返回受影响的行数

//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类似对象的形式组织的.
不是关联数组.

```php
// 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()
//如果你不需要一整行数据，则可以使用 value 方法来从单条记录中取出单个值。此方法将直接返回字段的值：
$email = DB::table('users')->where('name', 'John')->value('email');

//你可以通过pluck方法获取一列的单个字段
$titles = DB::table('roles')->pluck('title');

foreach ($titles as $title) {
    echo $title;
}
```

获取使用原生的sql语句：DB::select ('select * from goods');

等价于：limit 1;

value()和pluck()方法有利于数据库的优化，用多少数据取多少数据

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：

```php
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

```php
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

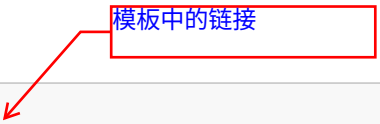
```
public function index() {
    $msgs = DB::table('msgs')->get();
    return view('msg.index' , ['msgs'=>$msgs]);
}
```

/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]);
    }
}
```

<project>/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('name'=>'MrChi');
$smarty->assign('problem'=>'what's wrong with you!');
```

在 blade 模板中,有着laravel独特的解析方法,例:

该视图文件位于 resources/views/greeting.blade.php，使用全局辅助函数 view 来返回：

```
Route::get('/', function () {
    return view('greeting', ['name' => 'MrChi']);
});
```

视图包含应用程序的 HTML，并且将控制器 / 应用程序逻辑与演示逻辑分开。视图文件存放于 resources/views 目录下。一个简单的视图如下所示：

```
<!-- 此视图文件位置：resources/views/greeting.blade.php -->

<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>
```

8.1.1模板赋值常用的几种方法

> **1)** 调用view函数,使用第二个参数进行赋值到模板,
> 注意:赋值一维数组和对象,在模板中取值的方式是不同的
> **2)** 调用with方法在view参数后使用
> 注意:`view('index')->with('name','Mrchi')`,模板中取`{{$name}}`,如:

```
Route::get('admin/index',function(){
    $data = array('title'=>'whoops : the page is not found');
    return view('index')->with('data',$data);
});

/*Blade模板使用*/
<body>
 <h1>Laravel MrChi</h1>
 <p>{{$data['title']}}</p>
 <p>give me a surprise , what's wrong with you thanks!</p>
</body>
```

3) 在`view`方法第二个参数使用`compact`方法
注意: `view('index',compact('data'))`,模板中取值如上

8.1.2 视图路由(5.5新增)

> 如果你的路由只需要返回一个视图，可以使用 `Route::view` 方法。
> view 方法有三个参数，其中前两个是必填参数，分别是 URL 和视图名称。第三个参数选填，可以传入一个数组，数组中的数据会被传递给视图

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

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,并且我们还可以使用isset和empty类似于php中直接在模板中判断

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

@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
```

我们还可以使用`switch case`这种方式,如下
`switch` 语句可以通过`@switch`，`@case`，`@break`，`@default` 和 `@enswitch` 指令构建：

```
@switch($i)
    @case(1)
        First case...
        @break

    @case(2)
        Second case...
        @break

    @default
        Default case...
@endswitch
```

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
```

在某些情况下，将 PHP 代码嵌入到视图中很有用。你可以使用 Blade 的 @php 指令在模板中执行一段纯 PHP 代码

```
@php

@endphp
```

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>
     <head>
         <title>Hello , @yield('title')</title>
     </head>
     <body>
         <div>
             @section('right')
             <h3><p>this is parent's body,please rewrite it!</p></h3>
             @show
         </div>

         <div class="container">
             @yield('content')
         </div>
     </body>
 </html>
```

如你所见，该文件包含了典型的 HTML 语法。不过，请注意 @Section 和 @yield 命令。顾名思义，@Section 命令定义了视图的一部分内容，而 @yield 指令是用来显示指定部分的内容。

父模板定义right方法,字模板继承right并重写方法

在这里@yield只是占位的作用

```
<!-- 子模板 child.blade.php-->
@extends('parent')

@section('title','MrChi Blog')

@section('right')
  <h2><p>this is mrchi's blog,thanks to give me a active!</p></h2>
@endsection

@section('content')
  <p>hello mrchi:I want to tell you ,this is my country!</p>
@endsection
```

@yield指令

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.

组件&插槽

不太懂

组件和插槽给内容片段（section）和布局（layout）带来了方便，不过，有些人可能会发现组件和插槽的模型更容易理解。首先，我们假设有一个可复用的“alert”组件，我们想要在整个应用中都可以复用它：

```
//alert.blade.php

<div class="alert alert-danger">
    <div class="alert-title">{{ $title }}</div>
    {{ $slot }}
</div>
```

`{{ $slot }}` 变量包含了我们想要注入组件的内容，现在，要构建这个组件，我们可以使用 Blade 指令 @component：

```
//使用组件
@component('alert')
    @slot('title')
        Forbidden
    @endslot

    You are not allowed to access this resource!
@endcomponent
```

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.

表名一般用复数形式，然后表名去掉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
```

直接调用，没有实例化

```
// 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：

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:

```php
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

```php
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

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

第10章 资源控制器

由于在laravel里声明路由,一条路由针对一个请求,使用起来比较麻烦,那么有没有一种办法让我们创建一条路由使用更多的方法去响应呢?如果是这样会更加的简单,那就是资源控制器.

资源控制器让你可以轻松地创建与资源相关的 RESTful 控制器。例如，你可能想要创建一个用来处理应用程序保存「文章」时发送 HTTP 请求的控制器。使用 `make:controller Artisan`命令，我们可以快速地创建一个像这样的控制器：

```
/***
*创建一个laravel帮助我们准备好方法的控制器
*/
php artisan make:controller ArticleController --resource

/***
*创建一个没有方法的控制器
*/
php artisan make:controller ArticleController
```

下面主要是对第一条artisan命令的阐述:
此 Artisan 命令会生成 `app/Http/Controllers/Article/Controller.php` 控制器文件。此控制器会包含用来操作可获取到的各种资源的方法,每个方法都有着不同的请求方式,比如get
,post,put,delete等,如:

```php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller;

class ArticleController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }
```

```php
    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
}
```

然后准备资源控制器路由

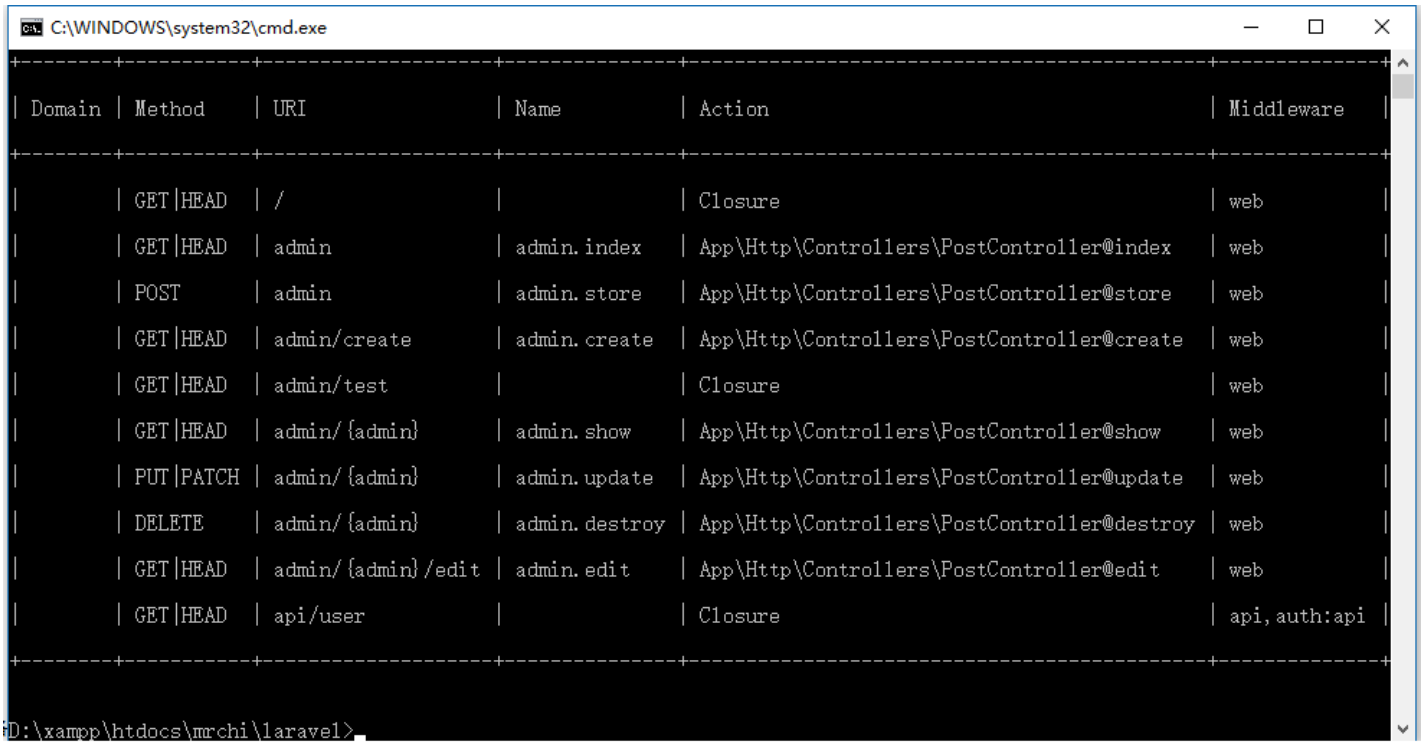
```
Route::resource('article','ArticleController');
```

可能会有同学有疑问,做完这些就完事了?是的,但是怎么去响应给我们准备好的方法呢?
这一条路由声明会创建多个路由，用来处理各式各样和相片资源相关的的 RESTful 行为。同样地，生成的控制器有着各种和这些行为绑定的方法，包含要处理的 URI 及方法对应的注释。

揭晓谜底,在cmd中运行命令

```
php artisan route:list
```

你会发现会多出很多路由,并且每条路由和我们生成的控制器中的方法相对应,如:



第11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`.

11.1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);
```

11.2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

11.3 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12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

12.1 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.
}
```

12.2 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)
```

12.3 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');
}
```

或者利用laravel提供的文件存储系统:
配置文件在`config/filesystems.php`,默认使用本地驱动,

```
    'disks' => [

        'local' => [
            'driver' => 'local',
            'root' => storage_path('app'),
        ],

        'public' => [
            'driver' => 'local', //本地驱动
            'root' => storage_path('app/public'), //存储的路径
            'url' => env('APP_URL').'/storage', //默认存储的文件夹
            'visibility' => 'public', //文件的名字
        ],
  //自己设置
        'uploads' => [
            'driver' => 'local',
            'root' => storage_path('app/uploads'),
        ],
```

当你使用驱动的时候,那么文件上传的方式是多种实现,如

```

        $path = $request->file('pic')->store('uploads');
        或者
        $path = Storage::putFile('uploads', $request->file('pic'));
  或者
        $path = $request->file('pic')->storeAs(
            'public', 'c.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>
```

12.4 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；

13 章 helper 辅助函数

Laravel 包含一些多样化的 PHP 辅助函数。
https://d.laravel-china.org/docs/5.5/helpers

- 数组函数
- 字符串函数
- 路径函数
- URL 函数
- 其他杂项函数

array_collapse 函数将数组的每一个数组折成单个数组：

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

str_limit 函数限制字符串的字符数量。该函数接受一个字符串作为第一个参数，以及最大字符数量作为第二参数：

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

str_random 函数生成指定长度的随机字符串：

```
echo(str_random(40));
```

e 函数对指定字符串进行实体转义，相当于运行 htmlentities：

```
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

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]);
```

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_field 函数生成包含 CSRF 令牌内容的 HTML 表单隐藏字段。
在Blade模板中，可以直接替换隐藏表单使用

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

request() 得到当前的 request 对象

```
$req = request();
dd($req);
```

- 至此，laravel 的基本用法课程结束，接下来，利用laravel框架做一个项目，在项目过程中，逐步加入laravel的高级用法；