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

21 章 功能细化

21.1 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
```

或者我们还可以使用身份快捷认证,如:
`@auth`和 `@guest` 指令可以用来快速确定当前用户是否已通过身份验证，是否为访客：

```
//用户通过身份认证
@auth
   <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>
@endauth

//用户未通过身份认证
@guest
    <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>
@endguest
```

21.2 分页功能

> 有几种方法可以对项目进行分页。最简单的是在`查询语句构造器` 或 `Eloquent` 查询 中使用 paginate 方法。
> 代码如下:

```
//查询构造器
public function myzd() {
    //带有样式的分页
    $hks = DB::table('hks')->paginate(2);
    return view('myzd' , ['hks'=>$hks]);
```

```
}

or
//简单分页
$users = DB::table('users')->simplePaginate(15);

//model查询使用
$users = User::where('votes', '>', 100)->paginate(15);
or
$users = User::where('votes', '>', 100)->simplePaginate(15);
```

如何渲染样式在Blade模板上?

```
<style media="screen">
    .pagination li{
    float: left;
    margin: 1px;
    }
</style>

//在blade模板中渲染样式
{{$hks->links}}
```

附加参数到分页链接中
你可以使用`append`方法附加查询参数到分页链接中。例如，要附加 `sort=vote`s 到每个分页链接，你应该这样调用 `append` 方法：

```
{{ $hks->appends(['sort' => 'votes'])->links() }}
```

> 如果你希望附加「哈希片段」到分页器的链接中，你应该使用`fragment` 方法。例如，要附加 `#foo` 到每个分页链接的末尾，应该这样调用 `fragment` 方法：
> ```php
> {{ $users->fragment('foo')->links() }}

```
**自定义分页视图**
>在默认情况下，视图渲染显示的分页链接都兼容 `Bootstrap CSS` 框架。但是，如果你不使用` Bootstrap，`你可以随意定义你自己的视图去渲染这些链接。当在分页

```php
{{ $hks->links('view.name') }}

// Passing data to the view...
{{ $hks->links('view.name', ['foo' => 'bar']) }}
```

然而，自定义分页视图最简单的方法是通过 `vendor:publish` 命令将它们导出到你的 `resources/views/vendor` 目录：

```
php artisan vendor:publish --tag=laravel-pagination
```

这个命令将视图放置在 resources/views/vendor/pagination 目录中。这个目录下的 default.blade.php 文件对应于默认分页视图。你可以简单地编辑这个文件来修改分页的 HTML 。

21.3 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;
    }
});
```

第22章 自动验证 自动验证

22.1 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
验证规则为关联数组,语法如下:

```
[
'待验证字段'=>'验证规则',
]

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.
```

借款验证案例:

```
// file: <project>/app/Http/Controllers/ProController.php
public function jiePost(Request $req) {
    $this->validate($req , [

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

验证未通过的检测,以money为例

```
$errors->has('money'):如果存在，则验证未通过
{{old('money')}}：如果验证未通过，表单中的上次信息保留，输出上一次表单数据
{{$errors->first('money')}}：输出错误信息

<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 ($errors->has('money'))
<p style="display: block;" id="amountError" class="error">
//原生错误提示
{{$errors->first('money')}}
//也可以是我们自己写的错误提示
</p>
@endif
```

22.2 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|nullable',
        'mobile'=>'required|regex:/^1[3458]\d{9}$/'|nullable,
        ] ,
        [
```

```
        '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'
        ]
        );
}

or

$req->validate([

]);
```

模板中使用 `{{$errors->first('money')}}` 输出即可

22.3 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
> 主要感受错误回馈和数据验证接受，正常开发，没必要使用；

```
// 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();
}
```

22.3 表单授权验证

> 面对更复杂的验证情境中，你可以创建一个「表单请求」来处理更为复杂的逻辑。表单请求是包含验证逻辑的自定义请求类。可使用 `Artisan`命令 `make:request` 来创建表单请求类：

`php artisan make:request SendReuest`

执行命令之后会在`App\Http\Requests`文件下发现

```
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class SendRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
```

```
        return true;//默认false 修改成true
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'mobile'=>'required | regex:/1[3578]\d{9}/',//定义自己的规则
        ];
    }
    //你还可以通过定义messages方法来自定义错误信息
    public function messages(){
      return [
          'mobile.required'=>'必须填写手机号!!!',
          'mobile.regex'=>'请输入正确的手机号!!!',
      ];
    }
}
```

控制器中使用,需要引入自己创建的Request对象,与laravel自带的Request对象使用方法一致,只不过是加上了自己的验证规则

```
    use App\Http\Requests\SendRequest;

    public function store(SendRequest $request)
    {
        //当提交form表单时会直接经过自己定义的验证处理

        return 'ok';
    }
```

22.4 自定义验证规则

> `Laravel` 提供了许多有用的验证规则，同时也支持自定义规则。注册自定义验证规则的方法之一，就是使用规则对象。可以使用 `Artisan` 命令 `make:rule`来生成新的规则对象。接下来，让我们用这个命令生成一个验证字符串是大写的规则。Laravel 会将新的规则存放在 `app/Rules` 目录中

`php artisan make:rule MobilePost`

> 一旦创建了规则，我们就可以定义它的行为。规则对象包含两个方法： passes 和 message 。 passes 方法接收属性值和名称，并根据属性值是否符合规则而返回 true 或者 false。 message 应返回验证失败时应使用的验证错误消息：

```
<?php

namespace App\Rules;

use Illuminate\Contracts\Validation\Rule;

class MobilePost implements Rule
{
    /**
     * Create a new rule instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Determine if the validation rule passes.
     *
```

```
     * @param  string  $attribute
     * @param  mixed  $value
     * @return bool
     */
    public function passes($attribute, $value)
    {
        return preg_match('/1[3578]\d{9}/', $value);//定义规则
    }

    /**
     * Get the validation error message.
     *
     * @return string
     */
    public function message()
    {
        return 'The mobile is errors';//定义错误信息
    }
}
```

控制器中通过使用`validate`方法进行验证

```
    use Illuminate\Http\Request;

    public function store(Request $request)
    {
        $request->validate([
            'mobile'=>['required',new MobilePost],
        ]);

        return 'ok';
    }
```

23 章 artisan 工具

如下：生成一个grow 命令

`php artisan make:command Grow`

执行命令之后,你会在`App\Console\Commands`文件下多出`Grow.php`

由于laravel5.5考虑用户的使用方便我们不在需要去手动在Kernel.php文件中注册命令

源头就是以下代码直接会把我们创建的命令自动加载,这样的话当你使用多个自己的命令时,不需要在一个一个去手动注册提供了很大的便利

```
    //kernel.php
    protected function commands()
    {
        $this->load(__DIR__.'/Commands');

        require base_path('routes/console.php');
    }
```

然后在Grow.php中修改

`protected $signature = 'grow';`//命令名字

`protected $description = 'Send drip e-mails to a user';`//命令描述

通过执行`php artisan list`进行查看到自己创建的命令

然后我们可以在`handle`方法中定义我们的代码逻辑

你就可以执行 `php artisan grow`命令来运行你的逻辑

Win: 控制面板 -> 管理工具 -> 任务计划

将张利的代码复制到handle 方法中，避免地址栏访问，产生虚假涨利

创建任务：

创建基本任务

启动程序脚本： `php C:\xampp\htdoc\dd\artisan grow`

24章 验证码类

gregwar/captcha

> 由于使用第三方扩展包,我们需要到 `Packagist.org`上查找我们需要的类,并使用composer加载到我们的项目中
> ```php
> composer require gregwar/captcha

```
>然后阅读文档,查看使用方式,并结合自己的项目来书写自己的验证逻辑
这里以借款的form表单为例,如
准备控制器和路由

```php
php artisan make:controller CaptchaController
Route::get('captcha','CaptchaController@captcha');
```

控制器准备

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Gregwar\Captcha\CaptchaBuilder;//引入类
use Session;
class CaptchaController extends Controller
{
    public function captcha(){
      $builder = new CaptchaBuilder;//实例化
      $builder->build();//创建验证码
      Session::put('img_code',strtoupper( $builder->getPhrase()));//把值存到session中
      header('Content-type: image/jpeg');
      $builder->output();//输出到模板
    }

}
```

验证准备

```
   if(strtoupper($req->imgcode) !== $req->session()->get('img_code')){
     return back()->with('msg','验证码错误!!!!');
   }
```

模板中准备

```
<div class="iptBox">
    <span class="message">图形验证码</span>
    <input class="short" name="imgcode" id="imgcode" placeholder="请输入图形验证码" type="text">
    <img src="{{url('captcha')}}" name="imgc" id="imgc" alt="换一张" class="get-code" height="36" width="80"
    onclick="this.src='{{url('captcha')}}?'+ Math.random()" >
    @if(session('msg'))
    <p id="imgcodeError"  style="display:block;" class="error">{{session('msg')}}</p>
    @endif
</div>
```
```