

Laravel 大型项目系列教程（七）

扩展包和 Artisan 开发

一、前言

本节教程将讲解扩展包开发和 Artisan 扩展开发，并浏览不同分辨率下的自适应效果。

二、Let's go

1. 扩展包开发

在前面开发中，我们经常要用到通知，如修改用户信息时视图要写

```
@if (Session::has('message'))
    <div class="am-alert am-alert-{{ Session::get('message')['type'] }}" data-am-alert>
        <p>{{ Session::get('message')['content'] }}</p>
    </div>@endif
```

在业务逻辑代码中需要使用

```
return Redirect::route('user.edit', $id)->with('user', $user)->with('message', array('type' => 'success', 'content' => 'Modify successfully'));
```

现在我们这里实现一个简单的通知插件，先创建包：

```
$ php artisan workbench shiyanlou/notification --resources
```

这时会在项目根目录下多一个名为 `workbench` 的目录，里面存放的就是刚才创建的包，我们进入 `shiyanlou/notification` 目录 `src/Shiyanlou/Notification` 目录是所有 `class` 的主目录，包括 `ServiceProvider`。config、lang、migrations 和 views 目录，就如你所猜测，包含了你创建的包的相应资源。包可以包含这些资源中的任意几个，就像一个“常规”的应用。

修改下包里 `composer.json` 中的 `authors`：

```
"authors": [
    {
        "name": "shiyanlou",
        "email": "support@shiyanlou.com"
    }
]
```

在项目根目录下执行：

```
$ php artisan dump-autoload
```

然后我们在 `app/config/app.php` 中的 `providers` 中增加：

```
'Shiyanlou\Notification\NotificationServiceProvider',
```

这步做完后启动开发服务器：

```
$ php artisan serve
```

如果启动成功，就说明扩展包的基础就搭建完成了。

现在我们在 `src/Shiyanlou/Notification` 下创建一个名为 `Notification.php` 的文件，修改：

```
<?php namespace Shiyanlou\Notification;
use Illuminate\Session\Store as SessionStore;
class Notification {
    private $session = null;

    public function __construct(SessionStore $session)
    {
        $this->session = $session;
    }

    private function addMessage($type, $content)
    {
        $this->session->put('notification_message', '<div class="am-alert ' . $type . '" data-am-alert><p></p>' . $content . '</div>');
    }

    public function primary($content)
    {
        $this->addMessage('am-alert-primary', $content);
    }

    public function secondary($content)
    {
        $this->addMessage('am-alert-secondary', $content);
    }

    public function success($content)
    {
        $this->addMessage('am-alert-success', $content);
    }

    public function warning($content)
    {
        $this->addMessage('am-alert-warning', $content);
    }

    public function danger($content)
    {
        $this->addMessage('am-alert-danger', $content);
    }
}
```

```

    public function show()
    {
        echo $this->session->pull('notification_message', "");
    }
}

```

上面用到了 `Session`，`Session` 表示一次会话，就是从你打开浏览器窗口到关闭。

修改 `NotificationServiceProvider.php` 中的 `register()` 和 `provides()`：

```

public function register(){
    $this->app['notification'] = $this->app->share(function($app)
    {
        return new Notification($this->app['session.store']);
    });
}
public function provides(){
    return array('notification');
}

```

上面是向 `Ioc` 容器注册类。

然后在 `src/Shiyanlou/Notification` 下创建一个名为 `Facades` 的文件夹，在 `Facades` 目录下创建一个名为 `Notification.php` 的文件，修改：

```

<?php namespace Shiyanlou\Notification\Facades;
use Illuminate\Support\Facades\Facade;
class Notification extends Facade {
    protected static function getFacadeAccessor()
    {
        return 'notification';
    }
}

```

我们这里继承了 `Facade` 类，用 `Facades` 可以访问 `IoC` 容器中注册的类，有了 `IoC` 容器，我们可以在任何地方调用注册的类。

为了方便我们的使用，我们在 `app/config/app.php` 的 `aliases` 中增加一个别名：

```

'Notification' => 'Shiyanlou\Notification\Facades\Notification',

```

下面就来试试这个插件，把上面的

```

@if (Session::has('message'))
    <div class="am-alert am-alert-{{ Session::get('message')['type'] }}" data-am-alert>
        <p>{{ Session::get('message')['content'] }}</p>
    </div>@endif

```

替换成

```

{{ Notification::show() }}

```

把

```
return Redirect::route('user.edit', $id)->with('user', $user)->with('message', array('type' => 'success', 'content' => 'Modify successfully'));
```

替换成

```
Notification::success('Modify successfully');return Redirect::route('user.edit', $id);
```

现在修改用户信息后提示成功的信息就能方便地显示出来：

ShiYanLou Blog My Articles snowsnow ▾

Modify successfully

E-mail:

NickName:

OldPassword:

NewPassword:

ConfirmPassword:

Modify

© 2015 By www.shianlou.com

简单的[扩展包开发](#)就完成了。

2. Artisan 扩展开发

Artisan 是 Laravel 中自带的命令行工具的名称，它提供了一些开发过程中有用的命令。我们可以编写自己的 Artisan 命令完成特定的功能，这里举一个开发导出用户数据的命令。首先我们创建一个新的命令类：

```
$ php artisan command:make ExportUsersCommand
```

执行完后我们会发现在 `app/commands` 生成了一个 `ExportUsersCommand.php` 的文件，这个就是我们自定义的命令类，然后我们需要注册命令，在 `app/start/artisan.php` 中增加：

```
Artisan::add(new ExportUsersCommand);
```

下面编写 `ExportUsersCommand` 类,把 `$name` 的值改为 `export:users`,这个 `$name` 是命令的名称,把 `$description` 的值改为 `Export all users`,这个是命令的描述,然后添加一个获取用户数据的方法:

```
protected function getUsersData(){
    $users = User::all();
    foreach ($users as $user) {
        $output[] = [$user->id, $user->email, $user->nickname,
                    $user->is_admin, $user->block, $user->created
                    _at];
    }
    return $output;
}
```

然后编写 `getArguments()`和 `getOptions()`:

```
protected function getArguments(){
    return array(
        array('file', InputArgument::OPTIONAL, 'The output file path', null),
    );
}
protected function getOptions(){
    return array(
        array('headers', null, InputOption::VALUE_NONE, 'Display headers?', null),
    );
}
```

`getArguments` 与 `getOptions` 方法是用来接收要传入您的自定义命令的地方,这两个方法都会回传一组命令数组,并由数组清单所组成。

下面开始编写 `fire()`:

```
public function fire(){
    $output_path = $this->argument('file');
    $headers = ['ID', 'E-mail', 'NickName', 'is_admin', 'is_block', 'CreateDateTime'];
    $rows = $this->getUsersData();
    if ($output_path) {
        $handle = fopen($output_path, 'w');
        if ($this->option('headers')) {
            fputcsv($handle, $headers);
        }
        foreach ($rows as $row) {
            fputcsv($handle, $row);
        }
        fclose($handle);
        $this->info("Exported list to $output_path");
    } else {
        $table = $this->getHelperSet()->get('table');
        $table->setHeaders($headers)->setRows($rows);
    }
}
```

```
        $table->render($this->getOutput());  
    }  
}
```

当自定义命令被执行时，将会调用 `fire` 方法，你可以在此加入任何的逻辑判断。
现在就可以测试我们自己开发的命令了，先执行：

```
$ php artisan export:users
```

执行后会在命令行终端输出用户列表，我们试试导出到一个文件：

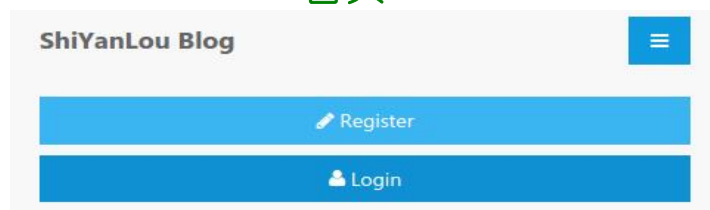
```
$ php artisan export:users --headers users.csv
```

执行后终端会输出 `Exported list to users.csv`，在项目根目录下会生成一个名为 `users.csv` 的文件，你可以用表格软件或者直接打开，里面存放的就是用户的数据列表。

3.自适应效果

让我们看下在低分辨率下的自适应效果

首页



The third article

by **admin** posted on 2015/01/26 14:11 under **IT**

1 This is a blog system developed by Laravel.



文章内容页面

2.Test

code:

```
function hello()  
{  
  return 'Welcome to ShiYanLou!';  
}
```

link:

www.shiyanlou.com

image:



登录页面

E-mail:

Password:

☐ Remember Me

Login

文章管理页面

ShiYanLou Blog				
Title	Tags	Author	Managment	
The fifth article	online	admin	 Edit	 Delete
The fourth article	online	snowsnow	 Edit	 Delete
The third article	IT	admin	 Edit	 Delete

© 2015 By www.shiyanlou.com

编辑文章页面


ShiYanLou Blog


Users


Articles

Tags

admin

 Publish Article

 Information

 Exit

My Articles



Preview

Tags

online

Separate multiple tags with a comma ","

Modify

© 2015 By www.shiyanlou.com

4.小结

本节教程介绍了怎么进行扩展包和 Artisan 开发，本套教程也就此结束了，你可以继续完善这个博客，此教程仅仅只是做一个引入人，你完全可以用 Laravel 开发自己想要的网站，Laravel 中的缓冲、Mail、本地化和队列等还没有提到，这就需要你自己去探索了，最后推荐一个开发环境 *Laravel Homestead*，我们可以非常方便地在其中开发 Laravel。

最终版代码下载：

```
$ git clone https://github.com/shiyanlou/laravel-blog-7-final.git
```

本文详细出处：<http://www.shiyanlou.com/courses/123>