Table of Contents

Universal 开发说明

环境配置

添加配置

存储在 .env 和 config/app.php 文件中，然后使用 config() 函数来读取

```
// .env
APP_PLATFORM=coffee

// config/app.php
return [
    ...

    'platform' => env('APP_PLATFORM')

    ...
]

//读取配置
config('app.platform')
```

配置缓存

配置信息缓存 生产环境中的 应该 使用『配置信息缓存』来加速 Laravel 配置信息的读取。

使用以下 Artisan 自带命令，把 config 文件夹里所有配置信息合并到一个文件里，减少运行时文件的载入数量：

```
php artisan config:cache
```

缓存文件存放在 bootstrap/cache/ 文件夹中。

可以使用以下命令来取消配置信息缓存：

```
php artisan config:clear
```

路由缓存

路由缓存 生产环境中的 应该 使用『路由缓存』来加速 Laravel 的路由注册。

路由缓存可以有效的提高路由器的注册效率，在大型应用程序中效果越加明显，可以使用以下命令

```
php artisan route:cache
```

缓存文件存放在 bootstrap/cache/ 文件夹中。另外，路由缓存不支持路由匿名函数编写逻辑

可以使用下面命令清除路由缓存：

```
php artisan route:clear
```

注意：路由缓存不会随着更新而自动重载，所以，开发时候建议关闭路由缓存，一般在生产环境中使用。可以配合 Envoy 任务运行器 使用，在每次上线代码时执行 route:clear 命令。

自动加载优化

此命令不止针对于 Laravel 程序，适用于所有使用 composer 来构建的程序。此命令会把 PSR-0 和 PSR-4 转换为一个类映射表，来提高类的加载速度。

```
composer dumpautoload
```

路由定义

遵循restful基础规则,尽量以名词定义一个资源(不绝对，比如signin，signup)，不了解的可自行搜索restful

restful的优点：

路由格式统一

面向资源,一目了然，自解释

方便前端调用

缺点：

过于理想化

实例

```
$api->group(["prefix" => "rooms"],function($api) {
    $api->get('/', 'RoomController@index');       // 获取room列表
    $api->post('/', 'RoomController@create');     // 创建room
    $api->get('/{id}', 'RoomController@view');    // 根据ID 获取
    $api->put('/{id}', 'RoomController@update');  // 根据ID 编辑
    $api->delete('/{id}', 'RoomController@delete'); // 根据ID 删
    $api->get('/{id}/machines', 'RoomController@machines');    /
    $api->get('/{id}/machines/{machineId}', 'RoomController@mach
});
```

禁用session和cookie

restful本身应为无状态请求，通过token授权，无需session和cookie，也不应该使用

laravel的API可能已经仅用了session和cookie的功能

响应

格式：

```
{
    "status_code": 200,     // 状态码
    "message": "success",   // 状态信息 对应上面的状态码
    "data":{                // 返回的数据，任何返回的数据都应该放在
    }
}
```

路由

路由表存放在routers目录下

每个项目单独建立路由表文件

```
routers
        coffee.php  //coffee项目路由表
        laundry.php //laundry项目路由表
```

代码会根据platform的配置会自动加载不同的路由表

```
// App\Providers\RouteServiceProvider

protected function getRoutersByPlatform($platform = 'coffee')
{
    switch ($platform) {
        case 'web':
            $path =  'routes/web.php';
            break;
        case 'laundry':
            $path =  'routes/laundry.php';
            break;
        default:
            $path =  'routes/coffee.php';
            break;
    }
    return $path;
}
```

控制器

目录

控制器根据当前项目放在不同目录下（子目录统一使用首字母大写）

```
Controllers\Api\V1
    Coffee      //coffee项目下所有controller目录
        Admin
        Kiosk
        Retemtion
    Laundry    //laundry项目下所有controller目录
        Admin

    ...        //其他项目
```

基类

controller需要尽量继承自App\Http\Controllers\Api\ApiController 或其子类

ApiController 注册了大部分业务所需要的中间件

```
class ApiController extends BaseController
{
    use MiddleWareTrait;

    public function __construct()
    {
        $this->registerMiddles();
    }
}
```

简单的**restful**控制器

只需要简单的CURD方法
可以继承 App\Http\Controllers\Api\RestController

```php
<?php

namespace App\Http\Controllers\Api\V1\Admin;


use App\Http\Controllers\Api\RestController;

class CoffeeMakeParamsController extends RestController
{
    public $modelClass = "App\Models\CoffeeMakeParams";
}
```

非**restful**控制器

不存在CURD方法，例如统计和报表

必须继承App\Http\Controllers\Api\ApiController

```php
<?php

namespace App\Http\Controllers\Api\V1\Admin;


use App\Http\Controllers\Api\ApiController;

class CoffeeMakeParamsController extends ApiController
{
    // TODO 你自己的业务代码

    ...
}
```

Model

所有的数据模型文件，都 必须 存放在：app/Models/ 文件夹中

所有的 Eloquent 数据模型 都 必须 继承统一的基类 App\Models\Model

非Eloquent 可以不继承

命名规范

- 数据模型类名 必须 为「单数」, 如：App\Models\Photo
- 类文件名 必须 为「单数」，如：app/Models/Photo.php
- 数据库表名字 必须 为「复数」，多个单词情况下使用「Snake Case」 如：photos, my_photos
- 数据库表迁移名字 必须 为「复数」，如：2014_08_08_234417_create_photos_table.php
- 数据填充文件名 必须 为「复数」，如：PhotosTableSeeder.php
- 数据库表主键 必须 为「id」
- 数据库表外键 必须 为「resource_id」，如：user_id, post_id
- 数据模型变量 必须 为「resource_id」，如：$user_id, $post_id

利用 **Trait** 来扩展数据模型

参考 validateTrait.php

时间格式

```
protected $dateFormat = 'U';  //使用unix 时间戳

protected $dateFormat = 'Y-m-d H:i:s';  //使用datetime格式
```

软删除

使用框架自带软删除

```
//引入软删除
use Illuminate\Database\Eloquent\SoftDeletes;

class Kiosk extends BaseModel
{
    use SoftDeletes;
}

//执行delete方法即可软删除
Kiosk::where('id',1)->first()->delete();
```

另外在使用模型relations时 需要特别注意软删除是否生效

迁移

- 使用 数据库迁移 去创建表结构，并提交版本控制器中
- 所有修改都 必须 使用 数据库迁移 ，并提交版本控制器中
- 所有数据填充需要使用seed

将service做成服务

app\providers\AppServiceProvider.php

```
public function register()
{
    $this->app->singleton('service',Service::class);
}
```

编写service业务代码

1.必须继承 App\Services\BaseService

2.公共service写在App\Services目录下

3.特殊service写在App\Services\{topic}\下,例如

App\Services\Coffee\OrderService.php
或者

App\Services\payment\Ibx.php

在控制器中使用service

```
class TestController extends BaseController
{
    public function test()
    {
        // 方式1
        app('service')->get('Location')->create();

        // 方式2
        app('service')->location->create();

        //手动指定目录
        app('service')->setTocpic('payment')->ibx->bind($params)
    }
}
```

Library

工具类存放在App\Libraries目录下

如何区分Library 和 service

独立的工具类，不涉及任何业务逻辑，可以使用在任何场景和项目中时，定义为Library

涉及业务逻辑，仅适用与当前项目的类，定义为service

例如

Libraries\lbx 和 services\payment\lbx

权限

角色

所有用户公用一张users表，使用role字段区分角色

```
define('ADMIN', 1);
define('RESIDENTS', 0);
```

JWT认证

```
// config/auth.php
'guards' => [
    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
        'hash' => false,
    ],

    'admin' => [
        'driver' => 'jwt',
        'provider' => 'admins',
        'hash' => false,
    ],
],

// User Model
class User extends Authenticatable implements JWTSubject
{
    ...

    protected $table = "users";

    public function getJWTCustomClaims()
    {
        return ['role' => 'user'];
    }
}

// Admin Model
class AdminUser extends \App\User
{
    protected static function booted()
    {
        static::addGlobalScope('role', function (Builder $builde
            $builder->where('role', ADMIN);
        });
    }

    public function getJWTCustomClaims()
    {
        return ['role' => 'admin'];
    }
}
```

不验证**auth**

具体实现可以参考 \App\Http\Controllers\Api\MiddleWareTrait

```
class TestController extends ApiController
{
    protected $authType = 'admin';

    protected $except =  ['login'];  // login方法不经过jwt的验证

    public function login();
}
```

常量定义

composer.json

```
"autoload": {
    "files": [
        "app/helpers.php",
        "app/common/constants.php"    //引入常量文件
    ]
}
```

constants.php

```
<?php
define('LAUNDRY', 'laundry');
define('COFFEE', 'coffee');
define('RETAIL', 'retail');
define('VENDING', 'vending');
define('INTEGRATION', 'integration');

define('ADMIN', 'admin');
define('SUPER-ADMIN', 'superAdmin');
define('CUSTOMER', 'customer');
```

使用

```
echo(SUPER-ADMIN);   //'superAdmin'
```

常量作用全局，即整个项目都可以使用，类常量表示该常量仅作用明确的类

类常量比较多且复杂时尽量写注释方便别人理解

```
const TYPE_REGULAR = 1;
const TYPE_SINGLE = 2;
```

全局方法

app\helpers.php

```
function respondWithToken($token, $data = [])
{
    $response = [
        'status_code' => ResponseCode::HTTP_OK,
        'access_token' => $token,
        'token_type' => 'bearer',
        'expires_in' => auth('api')->factory()->getTTL() * 60,
    ];
    if(!empty($data)) $response['data'] = $data;

    return response()->json($response);
}
```

phpUnit

laravel已自带 无需安装

配置

- phpunit配置文件为./phpunit.xml

添加测试套件

```
<testsuites>
    <testsuite name="Unit">
        <directory suffix="Test.php">./tests/Unit</directory>
    </testsuite>
    <testsuite name="Feature">
        <directory suffix="Test.php">./tests/Feature</directory>
    </testsuite>
    <testsuite name="Coffee">
        <directory suffix="Test.php">./tests/Coffee</directory>
    </testsuite>
    <testsuite name="Laundry">
        <directory suffix="Test.php">./tests/Laundry</directory>
    </testsuite>
    //TODO 添加其他测试套件

</testsuites>
```

- 新建testing数据库 并复制.env文件为.env.testing ,修改数据库信息为testing数据库

编写测试文件

测试代码存放在tests目录下

```
tests
    Coffee              //coffee项目测试目录
        Feature         //coffee项目功能测试目录
        Unit            //coffee项目单元测试目录
    Laundry             //laundry项目测试目录
        Feature         //laundry项目功能测试目录
        Unit            //laundry项目单元测试目录
    Feature             //通用功能测试目录
    Unit                //通用单元测试目录
```

代码量少且独立的代码使用单元测试，业务为主且逻辑复杂的代码使用功能测试

Feature test 示例

```
namespace Tests\Coffee\Feature\Admin;

use Tests\Laundry\TestRestCase;
use App\Models\Branch;

class BranchTest extends TestRestCase     // RestCase 简单封装了常
{

    public $source = '/api/admin/branches';   // restful 资源定义

    public function setUp() :void
    {
        parent::setUp();
        $this->generateToken();                 // 生成请求token
    }

    /**
     * @return void
     */
    public function testIndex() :void
    {
        factory(Branch::class,20)->create();            // 使用fa
        $response = $this->withToken($this->token)->json('GET',
        $response->assertStatus(200);                    // 验证
        $response->assertJsonPath('status_code', 200);
        $response->assertJsonStructure([
            'message',
            'status_code',
            'data' => [

            ]
        ]);
    }

    //TODO  其他测试
}
```

Unit test 示例

```
namespace Tests\Unit;

use Tests\TestCase;
use App\Libraries\Osoap;

class OsoapTest extends TestCase
{
    public function testdoAuth()         // 测试osoap绑卡功能
    {
        $params = [
            'firstname'  => 'ray',
            'lastname'   => 'ray',
            'exp_month' => '12',
            'exp_year'  => '2023',
            'card_number' => '1111111111111111',
            'cid' => '123'
        ];
        $response = $this->requestDoAuth($params);
        $this->assertArrayHasKey('error', $response);
        $this->assertTrue($response['error']);
    }
}
```

使用**setUp** 和 **factory**建造基境

每一个测试类的测试方法都会执行setUp方法，因此可以使用setUp方法为当前测试建立基境

使用factory可以为数据库提供模拟数据，以此模拟真正的数据环境

```
public function setUp() :void
{
    parent::setUp();
    $this->generateToken();                // 生成请求token
}
```

创建**factory**模型工厂

```
php artisan make:factory AdminUserFactory --model=AdminUser
```

编写**factory**

```
$factory->define(AdminUser::class, function (Faker $faker) {
    return [
        'email' => $faker->unique()->safeEmail,
        'email_verified_at' => now(),
        'password' => bcrypt("123456"), // password
        'first_name' => $faker->name,
        'remember_token' => Str::random(10),
        'mobile_number' => rand(100000,999999),
        'role' => 1,
        'invite_code' => Str::random(6),
    ];
});
```

使用**factory**

factory建好后laravel会自动和对应的model建立联系

```
//创建1个admin
$user = factory(\App\Models\AdminUser::class)->create();

//创建100个admin
$user = factory(\App\Models\AdminUser::class,100)->create();
```

每次测试后重置数据库

```
use Illuminate\Foundation\Testing\RefreshDatabase;
class ExampleTest extends TestCase
{
    use RefreshDatabase;
}
```

执行测试

```
php .\artisan test  //测试全部功能

php .\artisan test .\tests\Unit 测试通用单元

php .\artisan test .\tests\Laundry 测试laundry功能
```

生成测试报告

```
./vendor/bin/phpunit
```

测试报告存放在 ./test-result/report目录下

直接打开index.html可查看