Laravel 5.1 LTS 速查表

查看全部

```
// 在版本 5.1.11 新添加，见 http://laravel-china.org/docs/5.1/5.1/authorization#creating-policies
php artisan make:policy PostPolicy
// 针对命令显示帮助信息
php artisan --help OR -h
// 抑制输出信息
php artisan --quiet OR -q
// 打印 Laravel 的版本信息
php artisan --version OR -V
// 不询问任何交互性的问题
php artisan --no-interaction OR -n
// 强制输出 ANSI 格式
php artisan --ansi
// 禁止输出 ANSI 格式
php artisan --no-ansi
// 显示当前命令行运行的环境
php artisan --env
// -v|vv|vvv 通过增加 v 的个数来控制命令行输出内容的详尽情况：1 个代表正常输出，2 个代表输出更多消息，3 个代表调试
php artisan --verbose
// 移除编译优化过的文件 (storage/frameworks/compiled.php)
php artisan clear-compiled
// 显示当前框架运行的环境
php artisan env
// 显示某个命令的帮助信息
php artisan help
// 显示所有可用的命令
php artisan list
// 进入应用交互模式
php artisan tinker
// 进入维护模式
php artisan down
// 退出维护模式
php artisan up
// 优化框架性能// --force    强制编译已写入文件 (storage/frameworks/compiled.php)// --psr 不对 Composer 的 dump-autoload 进行优化
php artisan optimize [--force] [--psr]
// 启动内置服务器
php artisan serve
```

```
// 更改默认端口
php artisan serve --port 8080
// 使其在本地服务器外也可正常工作
php artisan serve --host 0.0.0.0
// 更改应用命名空间
php artisan app:name namespace
// 清除过期的密码重置令牌
php artisan auth:clear-resets
// 清空应用缓存
php artisan cache:clear
// 创建缓存数据库表 migration
php artisan cache:table
// 合并所有的配置信息为一个，提高加载速度
php artisan config:cache
// 移除配置缓存文件
php artisan config:clear
// 程序内部调用 Artisan 命令
$exitCode = Artisan::call('config:cache');
// 运行所有的 seed 假数据生成类// --class       可以指定运行的类，默认是:
"DatabaseSeeder"// --database    可以指定数据库// --force       当处于生产环境时强制执行操
作
php artisan db:seed [--class["..."]] [--database["..."]] [--force]

// 基于注册的信息，生成遗漏的 events 和 handlers
php artisan event:generate

// 生成新的处理器类// --command       需要处理器处理的命令类名字
php artisan handler:command [--command="..."] name
// 创建一个新的时间处理器类// --event         需要处理器处理的事件类名字// --queued
需要处理器使用队列话处理的事件类名字
php artisan handler:event [--event="..."] [--queued] name

// 生成应用的 key（会覆盖）
php artisan key:generate

// 在默认情况下，这将创建未加入队列的自处理命令// 通过 --handler 标识来生成一个处理器，用
--queued 来使其入队列.
php artisan make:command [--handler] [--queued] name
// 创建一个新的 Artisan 命令//  --command      命令被调用的名称。(默认为: "command:name")
php artisan make:console [--command["..."]] name
// 创建一个新的资源控制器// --plain       生成一个空白的控制器类
php artisan make:controller [--plain] name
php artisan make:controller App\\Admin\\Http\\Controllers\\DashboardController
// 创建一个新的事件类
```

```
php artisan make:event name
// 创建一个新的中间件类
php artisan make:middleware name
// 创建一个新的迁移文件// --create     将被创建的数据表.// --table      将被迁移的数据表.
php artisan make:migration [--create[="..."]] [--table[="..."]] name
// 创建一个新的 Eloquent 模型类
php artisan make:model name
// 创建一个新的服务提供者类
php artisan make:provider name
// 创建一个新的表单请求类
php artisan make:request name
// 数据库迁移// --database   指定数据库连接（下同）// --force      当处于生产环境时强制执行，不询问（下同）// --path       指定单独迁移文件地址// --pretend    把将要运行的 SQL 语句打印出来（下同）// --seed       Seed 任务是否需要被重新运行（下同）
php artisan migrate [--database[="..."]] [--force] [--path[="..."]] [--pretend] [--seed]
// 创建迁移数据库表
php artisan migrate:install [--database[="..."]]
// 重置并重新运行所有的 migrations// --seeder     指定主 Seeder 的类名
php artisan migrate:refresh [--database[="..."]] [--force] [--seed] [--seeder[="..."]]
// 回滚所有的数据库迁移
php artisan migrate:reset [--database[="..."]] [--force] [--pretend]
// 回滚最最近一次运行的迁移任务
php artisan migrate:rollback [--database[="..."]] [--force] [--pretend]
// migrations 数据库表信息
php artisan migrate:status
// 为队列数据库表创建一个新的迁移
php artisan queue:table
// 监听指定的队列// --queue      被监听的队列// --delay      给执行失败的任务设置延时时间(默认为零: 0)// --memory     内存限制大小，单位为 MB (默认为: 128)// --timeout    指定任务运行超时秒数 (默认为: 60)// --sleep      等待检查队列任务的秒数 (默认为: 3)// --tries      任务记录失败重试次数 (默认为: 0)
php artisan queue:listen [--queue[="..."]] [--delay[="..."]] [--memory[="..."]] [--timeout[="..."]] [--sleep[="..."]] [--tries[="..."]] [connection]
// 查看所有执行失败的队列任务
php artisan queue:failed
// 为执行失败的数据表任务创建一个迁移
php artisan queue:failed-table
// 清除所有执行失败的队列任务
php artisan queue:flush
// 删除一个执行失败的队列任务
php artisan queue:forget
// 在当前的队列任务执行完毕后, 重启队列的守护进程
php artisan queue:restart
// 对指定 id 的执行失败的队列任务进行重试(id: 失败队列任务的 ID)
```

```
php artisan queue:retry id
// 指定订阅 Iron.io 队列的链接// queue: Iron.io 的队列名称.// url: 将被订阅的 URL.// --type 指定队列的推送类型.
php artisan queue:subscribe [--type[="..."]] queue url
// 处理下一个队列任务// --queue 被监听的队列// --daemon 在后台模式运行// --delay 给执行失败的任务设置延时时间 (默认为零: 0)// --force 强制在「维护模式下」运行// --memory 内存限制大小，单位为 MB (默认为: 128)// --sleep 当没有任务处于有效状态时, 设置其进入休眠的秒数 (默认为: 3)// --tries 任务记录失败重试次数 (默认为: 0)
php artisan queue:work [--queue[="..."]] [--daemon] [--delay[="..."]] [--force] [--memory[="..."]] [--sleep[="..."]] [--tries[="..."]] [connection]

// 生成路由缓存文件来提升路由效率
php artisan route:cache
// 移除路由缓存文件
php artisan route:clear
// 显示已注册过的路由
php artisan route:list

// 运行计划命令
php artisan schedule:run

// 为 session 数据表生成迁移文件
php artisan session:table

// 从 vendor 的扩展包中发布任何可发布的资源// --force 重写所有已存在的文件// --provider 指定你想要发布资源文件的服务提供者// --tag 指定你想要发布标记资源.
php artisan vendor:publish [--force] [--provider[="..."]] [--tag[="..."]]
php artisan tail [--path[="..."]] [--lines[="..."]] [connection]
```

```
Route::get('foo', function(){});
Route::get('foo', 'ControllerName@function');
Route::controller('foo', 'FooController');
```

资源路由

```
Route::resource('posts','PostsController');
// 资源路由器只允许指定动作通过
Route::resource('photo', 'PhotoController',['only' => ['index', 'show']]);
Route::resource('photo', 'PhotoController',['except' => ['update', 'destroy']]);
```

触发错误

```
App::abort(404);
$handler->missing(...) in ErrorServiceProvider::boot();
throw new NotFoundHttpException;
```

路由参数

```
Route::get('foo/{bar}', function($bar){});
Route::get('foo/{bar?}', function($bar = 'bar'){});
```

HTTP 请求方式

```
Route::any('foo', function(){});
Route::post('foo', function(){});
Route::put('foo', function(){});
Route::patch('foo', function(){});
Route::delete('foo', function(){});
// RESTful 资源控制器
Route::resource('foo', 'FooController');
// 为一个路由注册多种请求方式
Route::match(['get', 'post'], '/', function(){});
```

安全路由 (TBD)

```
Route::get('foo', array('https', function(){}));
```

路由约束

```
Route::get('foo/{bar}', function($bar){})
->where('bar', '[0-9]+');
Route::get('foo/{bar}/{baz}', function($bar, $baz){})
->where(array('bar' => '[0-9]+', 'baz' => '[A-Za-z]'))
```

```
// 设置一个可跨路由使用的模式
Route::pattern('bar', '[0-9]+')
```

HTTP 中间件

命名路由

```
Route::currentRouteName();
Route::get('foo/bar', array('as' => 'foobar', function(){}));
Route::get('user/profile', [
'as' => 'profile', 'uses' => 'UserController@showProfile'
]);
Route::get('user/profile', 'UserController@showProfile')->name('profile');
$url = route('profile');
$redirect = redirect()->route('profile');
```

路由前缀

```
Route::group(['prefix' => 'admin'], function()
{
Route::get('users', function(){
return 'Matches The "/admin/users" URL';
});
});
```

路由命名空间

```
// 此路由组将会传送 'Foo\Bar' 命名空间
Route::group(array('namespace' => 'Foo\Bar'), function(){})
```

子域名路由

```
// {sub} 将在闭包中被忽略
Route::group(array('domain' => '{sub}.example.com'), function(){});
```

```
$environment = app()->environment();
$environment = App::environment();
$environment = $app->environment();
// 判断当环境是否为 local
if ($app->environment('local')){}
// 判断当环境是否为 local 或 staging...
if ($app->environment('local', 'staging')){}
```

```
// 记录器提供了 7 种在 RFC 5424 标准内定义的记录等级:// debug, info, notice, warning,
error, critical, and alert.
```

```
Log::info('info');
Log::info('info',array('context'=>'additional info'));
Log::error('error');
Log::warning('warning');
// 获取 monolog 实例
Log::getMonolog();
// 添加监听器
Log::listen(function($level, $message, $context) {});
```

记录 SQL 查询语句

```
// 开启 log
DB::connection()->enableQueryLog();
// 获取已执行的查询数组
DB::getQueryLog();
```

```
URL::full();
URL::current();
URL::previous();
URL::to('foo/bar', $parameters, $secure);
URL::action('NewsController@item', ['id'=>123]);
// 需要在适当的命名空间内
URL::action('Auth\AuthController@logout');
URL::action('FooController@method', $parameters, $absolute);
URL::route('foo', $parameters, $absolute);
URL::secure('foo/bar', $parameters);
URL::asset('css/foo.css', $secure);
URL::secureAsset('css/foo.css');
URL::isValidUrl('http://example.com');
URL::getRequest();
URL::setRequest($request);
```

```
Event::fire('foo.bar', array($bar));
// 注册一个事件监听器.// void listen(string|array $events, mixed $listener, int
$priority)
Event::listen('App\Events\UserSignup', function($bar){});
Event::listen('foo.*', function($bar){});
Event::listen('foo.bar', 'FooHandler', 10);
Event::listen('foo.bar', 'BarHandler', 5);
// 你可以直接在处理逻辑中返回 false 来停止一个事件的传播.
```

```
Event::listen('foor.bar', function($event){ return false; });
Event::subscribe('UserEventHandler');
```

基本使用

```
DB::connection('connection_name');
// 运行数据库查询语句
$results = DB::select('select * from users where id = ?', [1]);
$results = DB::select('select * from users where id = :id', ['id' => 1]);
// 运行普通语句
DB::statement('drop table users');
// 监听查询事件
DB::listen(function($sql, $bindings, $time){ code_here; });
// 数据库事务处理
DB::transaction(function()
{
DB::table('users')->update(['votes' => 1]);
DB::table('posts')->delete();
});
DB::beginTransaction();
DB::rollBack();
DB::commit();
```

查询语句构造器

```
// 取得数据表的所有行
DB::table('name')->get();
// 取数据表的部分数据
DB::table('users')->chunk(100, function($users)
{
  foreach ($users as $user)
  {
      //
  }
});
// 取回数据表的第一条数据
$user = DB::table('users')->where('name', 'John')->first();
DB::table('name')->first();
// 从单行中取出单列数据
$name = DB::table('users')->where('name', 'John')->pluck('name');
DB::table('name')->pluck('column');
// 取多行数据的「列数据」数组
$roles = DB::table('roles')->lists('title');
```

```
$roles = DB::table('roles')->lists('title', 'name');
// 指定一个选择字句
$users = DB::table('users')->select('name', 'email')->get();
$users = DB::table('users')->distinct()->get();
$users = DB::table('users')->select('name as user_name')->get();
// 添加一个选择字句到一个已存在的查询语句中
$query = DB::table('users')->select('name');
$users = $query->addSelect('age')->get();
// 使用 Where 运算符
$users = DB::table('users')->where('votes', '>', 100)->get();
$users = DB::table('users')
              ->where('votes', '>', 100)
              ->orWhere('name', 'John')
              ->get();
$users = DB::table('users')
      ->whereBetween('votes', [1, 100])->get();
$users = DB::table('users')
      ->whereNotBetween('votes', [1, 100])->get();
$users = DB::table('users')
      ->whereIn('id', [1, 2, 3])->get();
$users = DB::table('users')
      ->whereNotIn('id', [1, 2, 3])->get();
$users = DB::table('users')
      ->whereNull('updated_at')->get();
DB::table('name')->whereNotNull('column')->get();
// 动态的 Where 字句
$admin = DB::table('users')->whereId(1)->first();
$john = DB::table('users')
      ->whereIdAndEmail(2, 'john@doe.com')
      ->first();
$jane = DB::table('users')
      ->whereNameOrAge('Jane', 22)
      ->first();
// Order By, Group By, 和 Having
$users = DB::table('users')
      ->orderBy('name', 'desc')
      ->groupBy('count')
      ->having('count', '>', 100)
      ->get();
DB::table('name')->orderBy('column')->get();
DB::table('name')->orderBy('column','desc')->get();
DB::table('name')->having('count', '>', 100)->get();
// 偏移 & 限制
$users = DB::table('users')->skip(10)->take(5)->get();
```

Joins

```
// 基本的 Join 声明语句
DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.id', 'contacts.phone', 'orders.price')
    ->get();
// Left Join 声明语句
DB::table('users')
->leftJoin('posts', 'users.id', '=', 'posts.user_id')
->get();
// select * from users where name = 'John' or (votes > 100 and title <> 'Admin')
DB::table('users')
    ->where('name', '=', 'John')
    ->orWhere(function($query)
    {
        $query->where('votes', '>', 100)
              ->where('title', '<>', 'Admin');
    })
    ->get();
```

聚合

```
$users = DB::table('users')->count();
$price = DB::table('orders')->max('price');
$price = DB::table('orders')->min('price');
$price = DB::table('orders')->avg('price');
$total = DB::table('users')->sum('votes');
```

原始表达句

```
$users = DB::table('users')
                    ->select(DB::raw('count(*) as user_count, status'))
                    ->where('status', '<>', 1)
                    ->groupBy('status')
                    ->get();
// 返回行
DB::select('select * from users where id = ?', array('value'));
DB::insert('insert into foo set bar=2');
DB::update('update foo set bar=2');
```

```
DB::delete('delete from bar');
// 返回 void
DB::statement('update foo set bar=2');
// 在声明语句中加入原始的表达式
DB::table('name')->select(DB::raw('count(*) as count, column2'))->get();
```

Inserts / Updates / Deletes / Unions / Pessimistic Locking

```
// 插入
DB::table('users')->insert(
  ['email' => 'john@example.com', 'votes' => 0]
);
$id = DB::table('users')->insertGetId(
  ['email' => 'john@example.com', 'votes' => 0]
);
DB::table('users')->insert([
  ['email' => 'taylor@example.com', 'votes' => 0],
  ['email' => 'dayle@example.com', 'votes' => 0]
]);
// 更新
DB::table('users')
          ->where('id', 1)
          ->update(['votes' => 1]);
DB::table('users')->increment('votes');
DB::table('users')->increment('votes', 5);
DB::table('users')->decrement('votes');
DB::table('users')->decrement('votes', 5);
DB::table('users')->increment('votes', 1, ['name' => 'John']);
// 删除
DB::table('users')->where('votes', '<', 100)->delete();
DB::table('users')->delete();
DB::table('users')->truncate();
// 集合// unionAll() 方法也是可供使用的，调用方式与 union 相似
$first = DB::table('users')->whereNull('first_name');
$users = DB::table('users')->whereNull('last_name')->union($first)->get();
// 消极锁
DB::table('users')->where('votes', '>', 100)->sharedLock()->get();
DB::table('users')->where('votes', '>', 100)->lockForUpdate()->get();
```

基础使用

```
// 定义一个 Eloquent 模型
class User extends Model {}
```

```
// 生成一个 Eloquent 模型
php artisan make:model User
// 指定一个自定义的数据表名称
class User extends Model {
  protected $table = 'my_users';
}
```

More

```
Model::create(array('key' => 'value'));
// 通过属性找到第一条相匹配的数据或创造一条新数据
Model::firstOrCreate(array('key' => 'value'));
// 通过属性找到第一条相匹配的数据或实例化一条新数据
Model::firstOrNew(array('key' => 'value'));
// 通过属性找到相匹配的数据并更新，如果不存在即创建
Model::updateOrCreate(array('search_key' => 'search_value'), array('key' => 'value'));
// 使用属性的数组来填充一个模型，用的时候要小心「Mass Assignment」安全问题 !
Model::fill($attributes);
Model::destroy(1);
Model::all();
Model::find(1);
// 使用双主键进行查找
Model::find(array('first', 'last'));
// 查找失败时抛出异常
Model::findOrFail(1);
// 使用双主键进行查找，失败时抛出异常
Model::findOrFail(array('first', 'last'));
Model::where('foo', '=', 'bar')->get();
Model::where('foo', '=', 'bar')->first();
Model::where('foo', '=', 'bar')->exists();
// 动态属性查找
Model::whereFoo('bar')->first();
// 查找失败时抛出异常
Model::where('foo', '=', 'bar')->firstOrFail();
Model::where('foo', '=', 'bar')->count();
Model::where('foo', '=', 'bar')->delete();
// 输出原始的查询语句
Model::where('foo', '=', 'bar')->toSql();
Model::whereRaw('foo = bar and cars = 2', array(20))->get();
Model::on('connection-name')->find(1);
Model::with('relation')->get();
Model::all()->take(10);
Model::all()->skip(10);
```

```
// 默认的 Eloquent 排序是上升排序
Model::all()->orderBy('column');
Model::all()->orderBy('column','desc');
```

软删除

```
Model::withTrashed()->where('cars', 2)->get();
// 在查询结果中包括带被软删除的模型
Model::withTrashed()->where('cars', 2)->restore();
Model::where('cars', 2)->forceDelete();
// 查找只带有软删除的模型
Model::onlyTrashed()->where('cars', 2)->get();
```

模型关联

```
// 一对一 - User::phone()
  return $this->hasOne('App\Phone', 'foreign_key', 'local_key');
// 一对一 - Phone::user(), 定义相对的关联
return $this->belongsTo('App\User', 'foreign_key', 'other_key');

// 一对多 - Post::comments()
return $this->hasMany('App\Comment', 'foreign_key', 'local_key');
//  一对多 - Comment::post()
return $this->belongsTo('App\Post', 'foreign_key', 'other_key');

// 多对多 - User::roles();
return $this->belongsToMany('App\Role', 'user_roles', 'user_id', 'role_id');
// 多对多 - Role::users();
return $this->belongsToMany('App\User');
// 多对多 - Retrieving Intermediate Table Columns
$role->pivot->created_at;
// 多对多 - 中介表字段
return $this->belongsToMany('App\Role')->withPivot('column1', 'column2');
// 多对多 - 自动维护 created_at 和 updated_at 时间戳
return $this->belongsToMany('App\Role')->withTimestamps();

// 远层一对多 - Country::posts(), 一个 Country 模型可能通过中介的 Users// 模型关联到多个
Posts 模型(User::country_id)
return $this->hasManyThrough('App\Post', 'App\User', 'country_id', 'user_id');

// 多态关联 - Photo::imageable()
return $this->morphTo();
// 多态关联 - Staff::photos()
```

```
return $this->morphMany('App\Photo', 'imageable');
// 多态关联 - Product::photos()
return $this->morphMany('App\Photo', 'imageable');
// 多态关联 - 在 AppServiceProvider 中注册你的「多态对照表」
Relation::morphMap([
    'Post' => App\Post::class,
    'Comment' => App\Comment::class,
]);

// 多态多对多关联 - 涉及数据库表: posts,videos,tags,taggables// Post::tags()
return $this->morphToMany('App\Tag', 'taggable');
// Video::tags()
return $this->morphToMany('App\Tag', 'taggable');
// Tag::posts()
return $this->morphedByMany('App\Post', 'taggable');
// Tag::videos()
return $this->morphedByMany('App\Video', 'taggable');

// 查找关联
$user->posts()->where('active', 1)->get();
// 获取所有至少有一篇评论的文章...
$posts = App\Post::has('comments')->get();
// 获取所有至少有三篇评论的文章...
$posts = Post::has('comments', '>=', 3)->get();
// 获取所有至少有一篇评论被评分的文章...
$posts = Post::has('comments.votes')->get();
// 获取所有至少有一篇评论相似于 foo% 的文章
$posts = Post::whereHas('comments', function ($query) {
    $query->where('content', 'like', 'foo%');
})->get();

// 预加载
$books = App\Book::with('author')->get();
$books = App\Book::with('author', 'publisher')->get();
$books = App\Book::with('author.contacts')->get();

// 延迟预加载
$books->load('author', 'publisher');

// 写入关联模型
$comment = new App\Comment(['message' => 'A new comment.']);
$post->comments()->save($comment);
// Save 与多对多关联
$post->comments()->saveMany([
```

```
    new App\Comment(['message' => 'A new comment.']),
    new App\Comment(['message' => 'Another comment.']),
]);
$post->comments()->create(['message' => 'A new comment.']);

// 更新「从属」关联
$user->account()->associate($account);
$user->save();
$user->account()->dissociate();
$user->save();

// 附加多对多关系
$user->roles()->attach($roleId);
$user->roles()->attach($roleId, ['expires' => $expires]);
// 从用户上移除单一身份...
$user->roles()->detach($roleId);
// 从用户上移除所有身份...
$user->roles()->detach();
$user->roles()->detach([1, 2, 3]);
$user->roles()->attach([1 => ['expires' => $expires], 2, 3]);

// 任何不在给定数组中的 IDs 将会从中介表中被删除。
$user->roles()->sync([1, 2, 3]);
// 你也可以传递中介表上该 IDs 额外的值：
$user->roles()->sync([1 => ['expires' => true], 2, 3]);
```

事件

```
Model::creating(function($model){});
Model::created(function($model){});
Model::updating(function($model){});
Model::updated(function($model){});
Model::saving(function($model){});
Model::saved(function($model){});
Model::deleting(function($model){});
Model::deleted(function($model){});
Model::observe(new FooObserver);
```

Eloquent 配置信息

```
// 关闭模型插入或更新操作引发的 「mass assignment」异常
Eloquent::unguard();
// 重新开启「mass assignment」异常抛出功能
Eloquent::reguard();
```

```
// 自动处理分页逻辑
Model::paginate(15);
Model::where('cars', 2)->paginate(15);
// 使用简单模板 - 只有 "上一页" 或 "下一页" 链接
Model::where('cars', 2)->simplePaginate(15);
// 手动分页
Paginator::make($items, $totalItems, $perPage);
// 在页面打印分页导航栏
$variable->links();
```

```
App::setLocale('en');
Lang::get('messages.welcome');
Lang::get('messages.welcome', array('foo' => 'Bar'));
Lang::has('messages.welcome');
Lang::choice('messages.apples', 10);
// Lang::get 的别名
trans('messages.welcome');
```

```
File::exists('path');
File::get('path');
File::getRemote('path');
// 获取文件内容
File::getRequire('path');
// 获取文件内容, 仅能引入一次
File::requireOnce('path');
// 将内容写入文件
File::put('path', 'contents');
// 将内容添加在文件原内容后
File::append('path', 'data');
// 通过给定的路径来删除文件
File::delete('path');
// 将文件移动到新目录下
File::move('path', 'target');
// Copy a file to a new location// 将文件复制到新目录下
File::copy('path', 'target');
// 从文件的路径地址提取文件的扩展
File::extension('path');
// 获取文件类型
```

```
File::type('path');
// 获取文件大小
File::size('path');
// 获取文件的最后修改时间
File::lastModified('path');
// 判断给定的路径是否是文件目录
File::isDirectory('directory');
// 判断给定的路径是否是可写入的
File::isWritable('path');
// 判断给定的路径是否是文件
File::isFile('file');
// 查找能被匹配到的路径名
File::glob($patterns, $flag);
// Get an array of all files in a directory.// 获取一个目录下的所有文件, 以数组类型返回
File::files('directory');
// 获取一个目录下的所有文件 (递归).
File::allFiles('directory');
// 获取一个目录内的目录
File::directories('directory');
// 创建一个目录
File::makeDirectory('path',  $mode = 0777, $recursive = false);
// 将文件夹从一个目录复制到另一个目录下
File::copyDirectory('directory', 'destination', $options = null);
// 递归式删除目录
File::deleteDirectory('directory', $preserve = false);
// 清空指定目录的所有文件和文件夹
File::cleanDirectory('directory');
```

Install and run

```
// 将其加入到 composer.json 并更新:
composer require "phpunit/phpunit:4.0.*"
// 运行测试 (在项目根目录下运行)
./vendor/bin/phpunit
```

Asserts

```
$this->assertTrue(true);
$this->assertEquals('foo', $bar);
$this->assertCount(1,$times);
$this->assertResponseOk();
$this->assertResponseStatus(403);
$this->assertRedirectedTo('foo');
```

```
$this->assertRedirectedToRoute('route.name');
$this->assertRedirectedToAction('Controller@method');
$this->assertViewHas('name');
$this->assertViewHas('age', $value);
$this->assertSessionHasErrors();
// 由单个 key 值来假定 session 有错误...
$this->assertSessionHasErrors('name');
// 由多个 key 值来假定 session 有错误...
$this->assertSessionHasErrors(array('name', 'age'));
$this->assertHasOldInput();
```

Calling routes

```
$response = $this->call($method, $uri, $parameters, $files, $server, $content);
$response = $this->callSecure('GET', 'foo/bar');
$this->session(['foo' => 'bar']);
$this->flushSession();
$this->seed();
$this->seed($connection);
```

Executing Commands

```
SSH::run(array $commands);
// 指定 remote, 否则将使用默认值
SSH::into($remote)->run(array $commands);
SSH::run(array $commands, function($line)
{
  echo $line.PHP_EOL;
});
```

任务

```
// 定义任务
SSH::define($taskName, array $commands);
// 执行任务
SSH::task($taskName, function($line)
{
  echo $line.PHP_EOL;
});
```

SFTP 上传

```
SSH::put($localFile, $remotePath);
SSH::putString($string, $remotePath);
```

```
// 创建指定数据表
Schema::create('table', function($table)
{
  $table->increments('id');
});
// 指定一个连接
Schema::connection('foo')->create('table', function($table){});
// 通过给定的名称来重命名数据表
Schema::rename($from, $to);
// 移除指定数据表
Schema::drop('table');
// 当数据表存在时，将指定数据表移除
Schema::dropIfExists('table');
// 判断数据表是否存在
Schema::hasTable('table');
// 判断数据表是否有该列
Schema::hasColumn('table', 'column');
// 更新一个已存在的数据表
Schema::table('table', function($table){});
// 重命名数据表的列
$table->renameColumn('from', 'to');
// 移除指定的数据表列
$table->dropColumn(string|array);
// 指定数据表使用的存储引擎
$table->engine = 'InnoDB';
// 字段顺序，只能在 MySQL 中才能用
$table->string('name')->after('email');
```

索引

```
$table->string('column')->unique();
$table->primary('column');
// 创建一个双主键
$table->primary(array('first', 'last'));
$table->unique('column');
$table->unique('column', 'key_name');
// 创建一个双唯一性索引
$table->unique(array('first', 'last'));
```

```
$table->unique(array('first', 'last'), 'key_name');
$table->index('column');
$table->index('column', 'key_name');
// 创建一个双索引
$table->index(array('first', 'last'));
$table->index(array('first', 'last'), 'key_name');
$table->dropPrimary(array('column'));
$table->dropPrimary('table_column_primary');
$table->dropUnique(array('column'));
$table->dropUnique('table_column_unique');
$table->dropIndex(array('column'));
$table->dropIndex('table_column_index');
```

外键

```
$table->foreign('user_id')->references('id')->on('users');
$table->foreign('user_id')->references('id')->on('users')-
>onDelete('cascade'|'restrict'|'set null'|'no action');
$table->foreign('user_id')->references('id')->on('users')-
>onUpdate('cascade'|'restrict'|'set null'|'no action');
$table->dropForeign(array('user_id'));
$table->dropForeign('posts_user_id_foreign');
```

字段类型

```
// 自增
$table->increments('id');
$table->bigIncrements('id');

// 数字
$table->integer('votes');
$table->tinyInteger('votes');
$table->smallInteger('votes');
$table->mediumInteger('votes');
$table->bigInteger('votes');
$table->float('amount');
$table->double('column', 15, 8);
$table->decimal('amount', 5, 2);

// 字符串和文本
$table->char('name', 4);
$table->string('email');
$table->string('name', 100);
```

```
$table->text('description');
$table->mediumText('description');
$table->longText('description');

// 日期和时间
$table->date('created_at');
$table->dateTime('created_at');
$table->time('sunrise');
$table->timestamp('added_on');
// Adds created_at and updated_at columns// 添加 created_at 和 updated_at 行
$table->timestamps();
$table->nullableTimestamps();

// 其它类型
$table->binary('data');
$table->boolean('confirmed');
// 为软删除添加 deleted_at 字段
$table->softDeletes();
$table->enum('choices', array('foo', 'bar'));
// 添加 remember_token 为 VARCHAR(100) NULL
$table->rememberToken();
// 添加整型的 parent_id 和字符串类型的 parent_type
$table->morphs('parent');
->nullable()
->default($value)
->unsigned()
```

```
Input::get('key');
// 指定默认值
Input::get('key', 'default');
Input::has('key');
Input::all();
// 只取回 'foo' 和 'bar', 返回数组
Input::only('foo', 'bar');
// 取除了 'foo' 的所有用户输入数组
Input::except('foo');
Input::flush();
```

会话周期内 Input

```
// 清除会话周期内的输入
Input::flash();
```

```
// 清除会话周期内的指定输入
Input::flashOnly('foo', 'bar');
// 清除会话周期内的除了指定的其他输入
Input::flashExcept('foo', 'baz');
// 取回一个旧的输入条目
Input::old('key','default_value');
```

Files

```
// 使用一个已上传的文件
Input::file('filename');
// 判断文件是否已上传
Input::hasFile('filename');
// 获取文件属性
Input::file('name')->getRealPath();
Input::file('name')->getClientOriginalName();
Input::file('name')->getClientOriginalExtension();
Input::file('name')->getSize();
Input::file('name')->getMimeType();
// 移动一个已上传的文件
Input::file('name')->move($destinationPath);
// 移动一个已上传的文件，并设置新的名字
Input::file('name')->move($destinationPath, $fileName);
```

```
Cache::put('key', 'value', $minutes);
Cache::add('key', 'value', $minutes);
Cache::forever('key', 'value');
Cache::remember('key', $minutes, function(){ return 'value' });
Cache::rememberForever('key', function(){ return 'value' });
Cache::forget('key');
Cache::has('key');
Cache::get('key');
Cache::get('key', 'default');
Cache::get('key', function(){ return 'default'; });
Cache::tags('my-tag')->put('key','value', $minutes);
Cache::tags('my-tag')->has('key');
Cache::tags('my-tag')->get('key');
Cache::tags('my-tag')->forget('key');
Cache::tags('my-tag')->flush();
Cache::increment('key');
Cache::increment('key', $amount);
Cache::decrement('key');
```

```
Cache::decrement('key', $amount);
Cache::section('group')->put('key', $value);
Cache::section('group')->get('key');
Cache::section('group')->flush();
```

```
Cookie::get('key');
Cookie::get('key', 'default');
// 创建一个永久有效的 cookie
Cookie::forever('key', 'value');
// 创建一个 N 分钟有效的 cookie
Cookie::make('key', 'value', 'minutes');
// 在回应之前先积累 cookie，回应时统一返回
Cookie::queue('key', 'value', 'minutes');
// 移除 Cookie
Cookie::forget('key');
// 从 response 发送一个 cookie
$response = Response::make('Hello World');
$response->withCookie(Cookie::make('name', 'value', $minutes));
```

```
Session::get('key');
// 从会话中读取一个条目
Session::get('key', 'default');
Session::get('key', function(){ return 'default'; });
// 获取 session 的 ID
Session::getId();
// 增加一个会话键值数据
Session::put('key', 'value');
// 将一个值加入到 session 的数组中
Session::push('foo.bar','value');
// 返回 session 的所有条目
Session::all();
// 检查 session 里是否有此条目
Session::has('key');
// 从 session 中移除一个条目
Session::forget('key');
// 从 session 中移除所有条目
Session::flush();
// 生成一个新的 session 标识符
Session::regenerate();
// 把一条数据暂存到 session 中
Session::flash('key', 'value');
```

```
// 清空所有的暂存数据
Session::reflash();
// 重新暂存当前暂存数据的子集
Session::keep(array('key1', 'key2'));
```

```
// url: http://xx.com/aa/bb
Request::url();
// 路径: /aa/bb
Request::path();
// 获取请求 Uri: /aa/bb/?c=d
Request::getRequestUri();
// 返回用户的 IP
Request::ip();
// 获取 Uri: http://xx.com/aa/bb/?c=d
Request::getUri();
// 获取查询字符串: c=d
Request::getQueryString();
// 获取请求端口 (例如 80, 443 等等)
Request::getPort();
// 判断当前请求的 URI 是否可被匹配
Request::is('foo/*');
// 获取 URI 的分段值 (索引从 1 开始)
Request::segment(1);
// 从请求中取回头部信息
Request::header('Content-Type');
// 从请求中取回服务器变量
Request::server('PATH_INFO');
// 判断请求是否是 AJAX 请求
Request::ajax();
// 判断请求是否使用 HTTPS
Request::secure();
// 获取请求方法
Request::method();
// 判断请求方法是否是指定类型的
Request::isMethod('post');
// 获取原始的 POST 数据
Request::instance()->getContent();
// 获取请求要求返回的格式
Request::format();
// 判断 HTTP Content-Type 头部信息是否包含 */json
Request::isJson();
// 判断 HTTP Accept 头部信息是否为 application/json
Request::wantsJson();
```

```
return Response::make($contents);
return Response::make($contents, 200);
return Response::json(array('key' => 'value'));
return Response::json(array('key' => 'value'))
->setCallback(Input::get('callback'));
return Response::download($filepath);
return Response::download($filepath, $filename, $headers);
// 创建一个回应且修改其头部信息的值
$response = Response::make($contents, 200);
$response->header('Content-Type', 'application/json');
return $response;
// 为回应附加上 cookie
return Response::make($content)
->withCookie(Cookie::make('key', 'value'));
```

```
return Redirect::to('foo/bar');
return Redirect::to('foo/bar')->with('key', 'value');
return Redirect::to('foo/bar')->withInput(Input::get());
return Redirect::to('foo/bar')->withInput(Input::except('password'));
return Redirect::to('foo/bar')->withErrors($validator);
// 重定向到之前的请求
return Redirect::back();
// 重定向到命名路由（根据命名路由算出 URL）
return Redirect::route('foobar');
return Redirect::route('foobar', array('value'));
return Redirect::route('foobar', array('key' => 'value'));
// 重定向到控制器动作（根据控制器动作算出 URL）
return Redirect::action('FooController@index');
return Redirect::action('FooController@baz', array('value'));
return Redirect::action('FooController@baz', array('key' => 'value'));
// 跳转到目的地址，如果没有设置则使用默认值 foo/bar
return Redirect::intended('foo/bar');
```

```
App::bind('foo', function($app){ return new Foo; });
App::make('foo');
// 如果存在此类，则返回
App::make('FooBar');
// 单例模式实例到服务容器中
App::singleton('foo', function(){ return new Foo; });
```

```
// 将已实例化的对象注册到服务容器中
App::instance('foo', new Foo);
// 注册绑定规则到服务容器中
App::bind('FooRepositoryInterface', 'BarRepository');
// 给应用注册一个服务提供者
App::register('FooServiceProvider');
// 监听容器对某个对象的解析
App::resolving(function($object){});
```

哈希

```
Hash::make('secretpassword');
Hash::check('secretpassword', $hashedPassword);
Hash::needsRehash($hashedPassword);
```

加密解密

```
Crypt::encrypt('secretstring');
Crypt::decrypt($encryptedString);
Crypt::setMode('ctr');
Crypt::setCipher($cipher);
```

```
// 判断当前用户是否已认证（是否已登录）
Auth::check();
// 获取当前的认证用户
Auth::user();
// 获取当前的认证用户的 ID（未登录情况下会报错）
Auth::id();
// 通过给定的信息来尝试对用户进行认证（成功后会自动启动会话）
Auth::attempt(['email' => $email, 'password' => $password]);
// 通过 Auth::attempt() 传入 true 值来开启 '记住我' 功能
Auth::attempt($credentials, true);
// 只针对一次的请求来认证用户
Auth::once($credentials);
// 登录一个指定用户到应用上
Auth::login(User::find(1));
// 登录指定用户 ID 的用户到应用上
Auth::loginUsingId(1);
// 使用户退出登录（清除会话）
Auth::logout();
// 验证用户凭证
```

```
Auth::validate($credentials);
// Attempt to authenticate using HTTP Basic Auth// 使用 HTTP 的基本认证方式来认证
Auth::basic('username');
// Perform a stateless HTTP Basic login attempt// 执行「HTTP Basic」登录尝试
Auth::onceBasic();
// 发送密码重置提示给用户
Password::remind($credentials, function($message, $user){});
```

用户授权

```
// 定义权限
Gate::define('update-post', 'Class@method');
Gate::define('update-post', function ($user, $post) {...});
// 传递多个参数
Gate::define('delete-comment', function ($user, $post, $comment) {});

// 检查权限
Gate::denies('update-post', $post);
Gate::allows('update-post', $post);
Gate::check('update-post', $post);
// 指定用户进行检查
Gate::forUser($user)->allows('update-post', $post);
// 在 User 模型下，使用 Authorizable trait
User::find(1)->can('update-post', $post);
User::find(1)->cannot('update-post', $post);

// 拦截所有检查
Gate::before(function ($user, $ability) {});
Gate::after(function ($user, $ability) {});

// Blade 模板语法
@can('update-post', $post)
@endcan
// 支持 else 表达式
@can('update-post', $post)
@else
@endcan

// 生成一个新的策略
php artisan make:policy PostPolicy
// `policy` 帮助函数
policy($post)->update($user, $post)
```

```
// 控制器授权
$this->authorize('update', $post);
// 指定用户 $user 授权
$this->authorizeForUser($user, 'update', $post);
```

```
Mail::send('email.view', $data, function($message){});
Mail::send(array('html.view', 'text.view'), $data, $callback);
Mail::queue('email.view', $data, function($message){});
Mail::queueOn('queue-name', 'email.view', $data, $callback);
Mail::later(5, 'email.view', $data, function($message){});
// 零时将发送邮件请求写入 log，方便测试
Mail::pretend();
```

```
// 这些都能在 $message 实例中使用, 并可传入到 Mail::send() 或 Mail::queue()
$message->from('email@example.com', 'Mr. Example');
$message->sender('email@example.com', 'Mr. Example');
$message->returnPath('email@example.com');
$message->to('email@example.com', 'Mr. Example');
$message->cc('email@example.com', 'Mr. Example');
$message->bcc('email@example.com', 'Mr. Example');
$message->replyTo('email@example.com', 'Mr. Example');
$message->subject('Welcome to the Jungle');
$message->priority(2);
$message->attach('foo\bar.txt', $options);
// 使用内存数据作为附件
$message->attachData('bar', 'Data Name', $options);
// 附带文件，并返回 CID
$message->embed('foo\bar.txt');
$message->embedData('foo', 'Data Name', $options);
// 获取底层的 Swift Message 对象
$message->getSwiftMessage();
```

```
Queue::push('SendMail', array('message' => $message));
Queue::push('SendEmail@send', array('message' => $message));
Queue::push(function($job) use $id {});
// 在多个 workers 中使用相同的负载
Queue::bulk(array('SendEmail', 'NotifyUser'), $payload);
// 开启队列监听器
php artisan queue:listen
```

```
php artisan queue:listen connection
php artisan queue:listen --timeout=60
// 只处理第一个队列任务
php artisan queue:work
// 在后台模式启动一个队列 worker
php artisan queue:work --daemon
// 为失败的任务创建 migration 文件
php artisan queue:failed-table
// 监听失败任务
php artisan queue:failed
// 通过 id 删除失败的任务
php artisan queue:forget 5
// 删除所有失败任务
php artisan queue:flush
```

```
Validator::make(
array('key' => 'Foo'),
array('key' => 'required|in:Foo')
);
Validator::extend('foo', function($attribute, $value, $params){});
Validator::extend('foo', 'FooValidator@validate');
Validator::resolver(function($translator, $data, $rules, $msgs)
{
return new FooValidator($translator, $data, $rules, $msgs);
});
```

Rules

```
accepted
active_url
after:YYYY-MM-DD
before:YYYY-MM-DD
alpha
alpha_dash
alpha_num
array
between:1,10
confirmed
date
date_format:YYYY-MM-DD
different:fieldname
digits:value
```

```
digits_between:min,max
boolean
email
exists:table,column
image
in:foo,bar,...
not_in:foo,bar,...
integer
numeric
ip
max:value
min:value
mimes:jpeg,png
regex:[0-9]
required
required_if:field,value
required_with:foo,bar,...
required_with_all:foo,bar,...
required_without:foo,bar,...
required_without_all:foo,bar,...
same:field
size:value
timezone
unique:table,column,except,idColumn
url
```

```
View::make('path/to/view');
View::make('foo/bar')->with('key', 'value');
View::make('foo/bar')->withKey('value');
View::make('foo/bar', array('key' => 'value'));
View::exists('foo/bar');
// 跨视图共享变量
View::share('key', 'value');
// 视图嵌套
View::make('foo/bar')->nest('name', 'foo/baz', $data);
// 注册一个视图构造器
View::composer('viewname', function($view){});
// 注册多个视图到一个视图构造器中
View::composer(array('view1', 'view2'), function($view){});
// 注册一个视图构造器类
View::composer('viewname', 'FooComposer');
View::creator('viewname', function($view){});
```

```
// 区块占位
@yield('name')
// 扩展布局模板
@extends('layout.name')
// 实现命名为 name 的区块（yield 占位的地方）
@section('name')
@stop
// 可继承内容区块
@section('sidebar')
@show
// 继承父模板内容（@show 的区块内容）
@parent
// 包含子视图
@include('view.name')
// 包含子视图，并传参
@include('view.name', array('key' => 'value'));
// 加载本地化语句
@lang('messages.name')
@choice('messages.name', 1);

@if
@else
@elseif
@endif

@unless
@endunless

@for
@endfor

@foreach
@endforeach

@while
@endwhile

//forelse 4.2 feature
@forelse($users as $user)
@empty
@endforelse
```

```
// 输出内容，被转义过的
{{ $var }}
// 输出未转义内容，5.0 特性
{!! $var !!}
{{-- Blade 注释，不会被输出到页面中 --}}
// 三元表达式的简写，以下相当于「$name ? $name : 'Default'」
{{{ $name or 'Default' }}}
// 保留双大括号，以下会编译为 {{ name }}
@{{ name }}
```

```
Form::open(array('url' => 'foo/bar', 'method' => 'PUT'));
Form::open(array('route' => 'foo.bar'));
Form::open(array('route' => array('foo.bar', $parameter)));
Form::open(array('action' => 'FooController@method'));
Form::open(array('action' => array('FooController@method', $parameter)));
Form::open(array('url' => 'foo/bar', 'files' => true));
Form::close();
Form::token();
Form::model($foo, array('route' => array('foo.bar', $foo->bar)));
```

Form Elements

```
Form::label('id', 'Description');
Form::label('id', 'Description', array('class' => 'foo'));
Form::text('name');
Form::text('name', $value);
Form::text('name', $value, array('class' => 'name'));
Form::textarea('name');
Form::textarea('name', $value);
Form::textarea('name', $value, array('class' => 'name'));
Form::hidden('foo', $value);
Form::password('password');
Form::password('password', array('placeholder' => 'Password'));
Form::email('name', $value, array());
Form::file('name', array('class' => 'name'));
Form::checkbox('name', 'value');
// 生成一个被选中的复选框
Form::checkbox('name', 'value', true, array('class' => 'name'));
Form::radio('name', 'value');
// 生成一个被选中的单选框
Form::radio('name', 'value', true, array('class' => 'name'));
```

```
Form::select('name', array('key' => 'value'));
Form::select('name', array('key' => 'value'), 'key', array('class' => 'name'));
Form::selectRange('range', 1, 10);
Form::selectYear('year', 2011, 2015);
Form::selectMonth('month');
Form::submit('Submit!', array('class' => 'name'));
Form::button('name', array('class' => 'name'));
Form::macro('fooField', function()
{
return '<input type="custom"/>';
});
Form::fooField();
```

```
HTML::macro('name', function(){});
// 将 HTML 字符串转为实体
HTML::entities($value);
// 将实体转为 HTML 字符
HTML::decode($value);
// 生成 JavaScript 文件链接
HTML::script($url, $attributes);
// 生成 CSS 文件链接
HTML::style($url, $attributes);
// 生成一个 HTML 图片元素
HTML::image($url, $alt, $attributes);
// 生成一个 HTML 链接
HTML::link($url, 'title', $attributes, $secure);
// 生成一个 HTTPS 类型的 HTML 链接
HTML::secureLink($url, 'title', $attributes);
// 给资源文件生成一个 HTML link
HTML::linkAsset($url, 'title', $attributes, $secure);
// 给资源文件生成一个 HTTPS HTML 链接
HTML::linkSecureAsset($url, 'title', $attributes);
// 给命名路由生成一个 HTML 链接
HTML::linkRoute($name, 'title', $parameters, $attributes);
// 给控制器动作生成 HTML 链接
HTML::linkAction($action, 'title', $parameters, $attributes);
// 给邮件地址生成 HTML 链接
HTML::mailto($email, 'title', $attributes);
// 混淆一个邮件地址以阻止垃圾邮件扫描器的嗅探
HTML::email($email);
// 生成有序列表
HTML::ol($list, $attributes);
// 生成无序列表
```

```
HTML::ul($list, $attributes);
HTML::listing($type, $list, $attributes);
HTML::listingElement($key, $type, $value);
HTML::nestedListing($key, $type, $value);
// 从数组中构建 HTML 属性
HTML::attributes($attributes);
// 构建单属性元素
HTML::attributeElement($key, $value);
// 混淆一个字符串以阻止垃圾邮件扫描器的嗅探
HTML::obfuscate($value);
```

```
// 将 UTF-8 的值直译为 ASCII 类型的值
Str::ascii($value)
Str::camel($value)
Str::contains($haystack, $needle)
Str::endsWith($haystack, $needles)
Str::finish($value, $cap)
Str::is($pattern, $value)
Str::length($value)
Str::limit($value, $limit = 100, $end = '...')
Str::lower($value)
Str::words($value, $words = 100, $end = '...')
Str::plural($value, $count = 2)
// 生成更加真实的 "随机" 字母数字字符串.
Str::random($length = 16)
// 生成一个 "随机" 字母数字字符串.
Str::quickRandom($length = 16)
Str::upper($value)
Str::title($value)
Str::singular($value)
Str::slug($title, $separator = '-')
Str::snake($value, $delimiter = '_')
Str::startsWith($haystack, $needles)
Str::studly($value)
Str::macro($name, $macro)
```

```
// 如果给定的键不存在于该数组，array_add 函数将给定的键值对加到数组中
array_add($array, 'key', 'value');
// 将数组的每一个数组折成单一数组
array_collapse($array);
// 函数返回两个数组，一个包含原本数组的键，另一个包含原本数组的值
```

```
array_divide($array);
// 把多维数组扁平化成一维数组，并用「点」式语法表示深度
array_dot($array);
// 从数组移除给定的键值对
array_except($array, array('key'));
// 返回数组中第一个通过为真测试的元素
array_first($array, function($key, $value){}, $default);
// 将多维数组扁平化成一维// ['Joe', 'PHP', 'Ruby'];
array_flatten(['name' => 'Joe', 'languages' => ['PHP', 'Ruby']]);
// 以「点」式语法从深度嵌套数组移除给定的键值对
array_forget($array, 'foo');
array_forget($array, 'foo.bar');
// 使用「点」式语法从深度嵌套数组取回给定的值
array_get($array, 'foo', 'default');
array_get($array, 'foo.bar', 'default');
// 使用「点」式语法检查给定的项目是否存在于数组中
array_has($array, 'products.desk');
// 从数组返回给定的键值对
array_only($array, array('key'));
// 从数组拉出一列给定的键值对
array_pluck($array, 'key');
// 从数组移除并返回给定的键值对
array_pull($array, 'key');
// 使用「点」式语法在深度嵌套数组中写入值
array_set($array, 'key', 'value');
array_set($array, 'key.subkey', 'value');
// 借由给定闭包结果排序数组
array_sort($array, function(){});
// 使用 sort 函数递归排序数组
array_sort_recursive();
// 使用给定的闭包过滤数组
array_where();
// 返回给定数组的第一个元素
head($array);
// 返回给定数组的最后一个元素
last($array);
```

```
// 取得 app 文件夹的完整路径
app_path();
// 取得项目根目录的完整路径
base_path();
// 取得应用配置目录的完整路径
config_path();
```

```
// 取得应用数据库目录的完整路径
database_path();
// 取得加上版本号的 Elixir 文件路径
elixir();
// 取得 public 目录的完整路径
public_path();
// 取得 storage 目录的完整路径
storage_path();
```

```
// 将给定的字符串转换成 驼峰式命名
camel_case($value);
// 返回不包含命名空间的类名称
class_basename($class);
class_basename($object);
// 对给定字符串运行 htmlentities
e('<html>');
// 判断字符串开头是否为给定内容
starts_with('Foo bar.', 'Foo');
// 判断给定字符串结尾是否为指定内容
ends_with('Foo bar.', 'bar.');
// 将给定的字符串转换成 蛇形命名
snake_case('fooBar');
// 限制字符串的字符数量
str_limit();
// 判断给定字符串是否包含指定内容
str_contains('Hello foo bar.', 'foo');
// 添加给定内容到字符串结尾，foo/bar/
str_finish('foo/bar', '/');
// 判断给定的字符串与给定的格式是否符合
str_is('foo*', 'foobar');
// 转换字符串成复数形
str_plural('car');
// 产生给定长度的随机字符串
str_random(25);
// 转换字符串成单数形。该函数目前仅支持英文
str_singular('cars');
// 从给定字符串产生网址友善的「slug」
str_slug("Laravel 5 Framework", "-");
// 将给定字符串转换成「首字大写命名」: FooBar
studly_case('foo_bar');
// 根据你的本地化文件翻译给定的语句
trans('foo.bar');
// 根据后缀变化翻译给定的语句
```

```
trans_choice('foo.bar', $count);
```

```
// 产生给定控制器行为网址
action('FooController@method', $parameters);
// 根据目前请求的协定（HTTP 或 HTTPS）产生资源文件网址
asset('img/photo.jpg', $title, $attributes);
// 根据 HTTPS 产生资源文件网址
secure_asset('img/photo.jpg', $title, $attributes);
// 产生给定路由名称网址
route($route, $parameters, $absolute = true);
// 产生给定路径的完整网址
url('path', $parameters = array(), $secure = null);
```

Miscellaneous

```
// 返回一个认证器实例。你可以使用它取代 Auth facade
auth()->user();
// 产生一个重定向回应让用户回到之前的位置
back();
// 使用 Bcrypt 哈希给定的数值。你可以使用它替代 Hash facade
bcrypt('my-secret-password');
// 从给定的项目产生集合实例
collect(['taylor', 'abigail']);
// 取得设置选项的设置值
config('app.timezone', $default);
// 产生包含 CSRF 令牌内容的 HTML 表单隐藏字段
{!! csrf_field() !!}
// 取得当前 CSRF 令牌的内容
$token = csrf_token();
// 输出给定变量并结束脚本运行
dd($value);
// 取得环境变量值或返回默认值
$env = env('APP_ENV');
$env = env('APP_ENV', 'production');
// 配送给定事件到所属的侦听器
event(new UserRegistered($user));
// 根据给定类、名称以及总数产生模型工厂建构器
$user = factory(App\User::class)->make();
// 产生拟造 HTTP 表单动作内容的 HTML 表单隐藏字段
{!! method_field('delete') !!}
// 取得快闪到 session 的旧有输入数值
$value = old('value');
```

```
$value = old('value', 'default');
// 返回重定向器实例以进行 重定向
return redirect('/home');
// 取得目前的请求实例或输入的项目
$value = request('key', $default = null)
// 创建一个回应实例或获取一个回应工厂实例
return response('Hello World', 200, $headers);
// 可被用于取得或设置单一 session 内容
$value = session('key');
// 在没有传递参数时，将返回 session 实例
$value = session()->get('key');
session()->put('key', $value);
// 返回给定数值
value(function(){ return 'bar'; });
// 取得视图 实例
return view('auth.login');
// 返回给定的数值
$value = with(new Foo)->work();
```

```
// 创建集合
collect([1, 2, 3]);
// 返回该集合所代表的底层数组：
$collection->all();
// 返回集合中所有项目的平均值：
$collection->avg();
// 将集合拆成多个给定大小的较小集合：
$collection->chunk(4);
// 将多个数组组成的集合折成单一数组集合：
$collection->collapse();
// 用来判断该集合是否含有指定的项目：
$collection->contains('New York');
// 返回该集合内的项目总数：
$collection->count();
// 遍历集合中的项目，并将之传入给定的回调函数：
$collection = $collection->each(function ($item, $key) {
});
// 会创建一个包含第 n 个元素的新集合：
$collection->every(4);
// 传递偏移值作为第二个参数：
$collection->every(4, 1);
// 返回集合中排除指定键的所有项目：
$collection->except(['price', 'discount']);
// 以给定的回调函数筛选集合，只留下那些通过判断测试的项目：
```

```
$filtered = $collection->filter(function ($item) {
    return $item > 2;
});
// 返回集合中，第一个通过给定测试的元素：
collect([1, 2, 3, 4])->first(function ($key, $value) {
    return $value > 2;
});
// 将多维集合转为一维集合：
$flattened = $collection->flatten();
// 将集合中的键和对应的数值进行互换：
$flipped = $collection->flip();
// 以键自集合移除掉一个项目：
$collection->forget('name');
// 返回含有可以用来在给定页码显示项目的新集合：
$chunk = $collection->forPage(2, 3);
// 返回给定键的项目。如果该键不存在，则返回 null：
$value = $collection->get('name');
// 根据给定的键替集合内的项目分组：
$grouped = $collection->groupBy('account_id');
// 用来确认集合中是否含有给定的键：
$collection->has('email');
// 用来连接集合中的项目
$collection->implode('product', ', ');
// 移除任何给定数组或集合内所没有的数值：
$intersect = $collection->intersect(['Desk', 'Chair', 'Bookcase']);
// 假如集合是空的，isEmpty 方法会返回 true：
collect([])->isEmpty();
// 以给定键的值作为集合项目的键：
$keyed = $collection->keyBy('product_id');
// 传入回调函数，该函数会返回集合的键的值：
$keyed = $collection->keyBy(function ($item) {
    return strtoupper($item['product_id']);
});
// 返回该集合所有的键：
$keys = $collection->keys();
// 返回集合中，最后一个通过给定测试的元素：
$collection->last();
// 遍历整个集合并将每一个数值传入给定的回调函数：
$multiplied = $collection->map(function ($item, $key) {
    return $item * 2;
});
// 返回给定键的最大值：
$max = collect([['foo' => 10], ['foo' => 20]])->max('foo');
$max = collect([1, 2, 3, 4, 5])->max();
```

```
// 将给定的数组合并进集合：
$merged = $collection->merge(['price' => 100, 'discount' => false]);
// 返回给定键的最小值：
$min = collect([['foo' => 10], ['foo' => 20]])->min('foo');
$min = collect([1, 2, 3, 4, 5])->min();
// 返回集合中指定键的所有项目：
$filtered = $collection->only(['product_id', 'name']);
// 获取所有集合中给定键的值：
$plucked = $collection->pluck('name');
// 移除并返回集合最后一个项目：
$collection->pop();
// 在集合前面增加一个项目：
$collection->prepend(0);
// 传递第二个参数来设置前置项目的键：
$collection->prepend(0, 'zero');
// 以键从集合中移除并返回一个项目：
$collection->pull('name');
// 附加一个项目到集合后面：
$collection->push(5);
// put 在集合内设置一个给定键和数值：
$collection->put('price', 100);
// 从集合中随机返回一个项目：
$collection->random();
// 传入一个整数到 random。如果该整数大于 1，则会返回一个集合：
$random = $collection->random(3);
// 会将每次迭代的结果传入到下一次迭代：
$total = $collection->reduce(function ($carry, $item) {
    return $carry + $item;
});
// 以给定的回调函数筛选集合：
$filtered = $collection->reject(function ($item) {
    return $item > 2;
});
// 反转集合内项目的顺序：
$reversed = $collection->reverse();
// 在集合内搜索给定的数值并返回找到的键：
$collection->search(4);
// 移除并返回集合的第一个项目：
$collection->shift();
// 随机排序集合的项目：
$shuffled = $collection->shuffle();
// 返回集合从给定索引开始的一部分切片：
$slice = $collection->slice(4);
// 对集合排序：
```

```
$sorted = $collection->sort();
// 以给定的键排序集合：
$sorted = $collection->sortBy('price');
// 移除并返回从指定的索引开始的一小切片项目：
$chunk = $collection->splice(2);
// 返回集合内所有项目的总和：
collect([1, 2, 3, 4, 5])->sum();
// 返回有着指定数量项目的集合：
$chunk = $collection->take(3);
// 将集合转换成纯 PHP 数组：
$collection->toArray();
// 将集合转换成 JSON：
$collection->toJson();
// 遍历集合并对集合内每一个项目调用给定的回调函数：
$collection->transform(function ($item, $key) {
    return $item * 2;
});
// 返回集合中所有唯一的项目：
$unique = $collection->unique();
// 返回键重设为连续整数的的新集合：
$values = $collection->values();
// 以一对给定的键 / 数值筛选集合：
$filtered = $collection->where('price', 100);
// 将集合与给定数组同样索引的值合并在一起：
$zipped = $collection->zip([100, 200]);
```