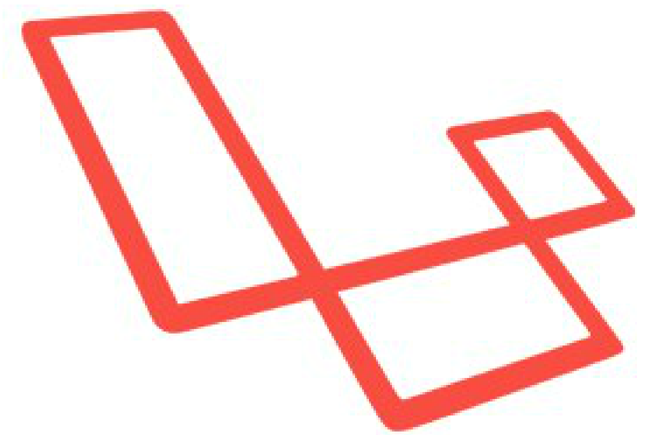


คู่มือการใช้งาน Laravel 4 ฉบับแปล

ก่อนอ่าน

เมื่อหลายเดือนก่อนผมพยายามมองหา framework ตัวใหม่ที่จะมาใช้แทน Cakephp มาเจอกับ Laravel เพราะเข้าไปอ่านบทความใน nettut บ่อยเลยลองใช้ดูแล้วพบว่า โค้ดจำนวนมากบท Cakephp สามารถย่อเหลือไม่กี่บรรทัดบน Laravel และที่สำคัญมากสำหรับผม การที่ไม่ต้องมาพิมพ์ `$this` ทุกครั้ง มันสุดยอดมาก ผมให้คะแนนเก้าเต็มสิบ สำหรับเรื่องความง่ายในการอ่านโค้ด ผมนึกถึงสมมุติถ้าเรานำเอาไปทำโปรเจคขนาดใหญ่ การที่โค้ดอ่านง่ายมาก แค่อ่านคู่มือเสร็จ แล้วมานั่งไล่โค้ดที่ได้ทำไว้แล้ว จะเป็นอะไรที่เรียนรู้ได้เร็วมาก แต่ข้อเสียของ laravel ก็คือไม่มีระบบอัติโนมัติมาให้ Route ก็ต้องเขียนเอง UI ก็ไม่มีตัวสร้างอัติโนมัติให้

เนื้อหาต่อไปนี้ผมได้ทำการแปล มาจากคู่มือหลักที่อยู่บนเว็บ ซึ่งทำการแปลตามคำอธิบาย และเสริมความเข้าใจตามที่ผมได้ทดลองทำมาหลายเดือน และบางหัวข้อ ในหนังสือนี้เป็นเรื่องหลักการของ php 5.3 ทั้งเรื่อง namespace,reflectionclass สำหรับคนที่ไม่เคยรู้มาก่อนอาจ จะงงเหมือนผม ในตอนแรก ซึ่งจะเขียนไว้ในนี้ก็จะยาวไป เพื่อการนั้นผมจึงได้ทำการอธิบายเพิ่มเติมไว้ที่ blog ของผม ในหนังสือนี้ อาจมีข้อผิดพลาดมากมาย ซึ่งผมอยากให้ใครที่เห็นข้อผิดพลาด ช่วยทำการแจ้งให้ผมทราบ หรือเข้าไปช่วยกันแก้ไขและเพิ่มเติมตัวอย่างการใช้งานฟังก์ชันต่างๆ ได้ที่ github

แนะนำตัวกันก่อน

ปรัชญาของ Laravel

Laravel เป็น php framework ที่เน้นไปที่ความเรียบง่ายของการใช้งานตัวแปรต่างๆ .
เราเชื่อว่าในการพัฒนาเว็บของคุณต้องเต็มไปด้วยความสนุกสนานแน่นอน,ประสบการณ์ความคิดสร้างสรรค์ที่จะตอบสนองความต้องการอย่างแท้จริง.
Laravel พยายามลดงานในระหว่างการพัฒนาโดยสร้างระบบสำเร็จรูปมาให้อย่าง authentication, routing, sessions,
และ caching.

Laravel ที่จะเป็นหนึ่งใน เครื่องมือในการพัฒนาที่นักพัฒนาชื่นชอบ
และสามารถใช้งานได้โดยที่ไม่ต้องเสียสละฟังก์ชันไหนไปเลย เราเชื่อว่า นักพัฒนาที่มีความสุขจะสร้างโค้ดที่เยี่ยมยอด
เราพยายามนำสิ่งที่ดีของภาษาอื่นๆ เช่น Ruby on Rails, ASP.NET MVC, และ Sinatra เข้ามาผสมผสานเข้ากับ
laravel

Laravel เตรียมเครื่องมือที่ใช้เพื่อการสร้างเว็บแอพลิเคชัน ที่ยืดหยุ่นด้วย inversion of control container,ระบบ
migration , และการทำ unit testing ที่แสนง่ายดาย

เรียนรู้ Laravel

หนึ่งในทางเลือกที่ดีคือการศึกษาคู่มือการใช้งาน หรือซื้อหนังสือในนี้ครับ

- Code Bright by Dayle Rees
- Laravel Testing Decoded by Jeffrey Way
- Laravel From Aprentice to Artisan by Taylor Otwell

นักพัฒนาหลัก

Laravel ถูกสร้างโดย Taylor Otwell, และ Dayle Rees, Shawn McCool, Jeffrey Way, Jason Lewis, Ben Corlett,
Franz Liedke, Dries Vints, Mior Muhammed Zaki, และ Phil Sturgeon.

ผู้สนับสนุน Laravel

- UserScape
- Cartalyst
- Elli Davis - Toronto Realtor
- Jay Banks - Vancouver Lofts & Condos
- Julie Kinnear - Toronto MLS
- Jamie Sarner - Toronto Real Estate

Installation

การติดตั้ง laravel

Install Composer

Laravel ใช้ Composer ในการจัดการไลบราลี่ต่างๆ รวมจนถึงคลาสหลักของระบบ เริ่มแรกเราต้องไปโหลด `composer.phar`. เราจะได้ไฟล์ที่มีนามสกุลเป็น phar มาแล้วเอาไปวางไว้ที่ `usr/local/bin` เพื่อให้ระบบมองเห็น บนวินโดเรามี Windows installer อยากรู้เพิ่มเติมเข้าไปดูได้บล็อกของผมได้ครับ

การติดตั้ง Laravel

ใช้ Composer ในการติดตั้ง

พิมพ์คำสั่งข้างล่างไปที่ commandline composer จะทำการดาวน์โหลดมาลงตรงที่เราเรียกใช้

```
composer create-project laravel/laravel
```

ดาวน์โหลดเอง

เมื่อติดตัง laravel เวอร์ชันล่าสุด แล้วก็แตกไฟล์ไปลงที่โฟลเดอร์ของ server เปิด command line เลือกที่อยู่ให้ตรงกับที่เอา laravel ไปวางแล้วรันคำสั่ง `php composer.phar install` (หรือ `composer install`) ก็เสร็จสิ้น

ถ้าเราต้องการอัพเดทก็ใช้คำสั่ง `php composer.phar update`

ความต้องการของระบบ

- PHP >= 5.3.7
- MCrypt PHP Extension

การตั้งค่า

laravel ไม่ได้ต้องการปรับแต่งอะไรมากเพียงแค่เราเข้าไปที่ `app/config/app.php` โดยอาจปรับแค่ `timezone` กับ `locale`

> **Note:** และมีค่า `key` ที่อยู่ใน `app/config/app.php`. เราต้องใช้คำสั่ง `php artisan key:generate`เพื่อสร้างคีย์ที่จะใช้สร้าง private key ในการสร้างรหัสผ่าน hash ในระบบ

สิทธิ

laravel ต้องการสิทธิในการอ่านเขียนโฟลเดอร์ app/storage

เส้นทาง

การกำหนดเส้นทางสามารถทำได้ที่ `bootstrap/paths.php`

URLs ที่สวยงาม

laravel เตรียมไฟล์ `public/.htaccess` ที่อนุญาตให้เราเรียกใช้งานโดยไม่ต้องใส่ `index.php`. โดยต้องการใช้งานขอ `mod_rewrite` บน server ก่อน

Laravel Lifecycle

ตอนที่ 1 Autoloading

มาดูวัฏจักรการทำงานของ Laravel กันครับ มีอยู่ด้วยกัน 4 ช่วง `Autoloading,Bootstrap,Application,Run` เริ่มที่ `Autoloading`ก่อนครับ

1. เริ่มจากไฟล์ `index.php` ครับ ช่วงที่เราตั้งค่า `vhost`เราจะตั้งให้เส้นทางที่เข้าถึงเริ่มแรก คือ `/public/` พอเริ่มเข้าถึงไฟล์ `index.php` ทำการเรียกไฟล์ `autoload.php` จากโฟลเดอร์ `bootstrap`
2. ในไฟล์ `autoload.php` จะทำการตั้งค่าเวลาที่ตัวเว็บเริ่มทำงาน

```
define('LARAVEL_START', microtime(true));
```

3. จากนั้นเราจะ เรียกไฟล์ `autoload.php` จากโฟลเดอร์ `vendor` ขึ้นมา

```
require __DIR__.'/../vendor/autoload.php';
```

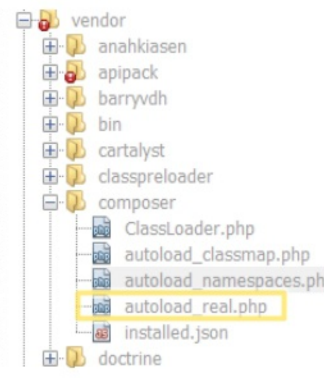
4. ที่เห็นชื่อยาวขนาดนี้เป็นเพราะถูกสุ่มมาจากการที่ใช้คำสั่ง`artisan dump autoload`เพื่อสร้าง รายชื่อของ `package` ใหม่นะครับ แต่ก่อนหน้านั้นเราจะทำการโหลดไฟล์ `package` ทั้งหมดก่อน โดยเรียกไฟล์ `autoload_real.php` ขึ้นมา

```php
require_once __DIR__ . '/composer' . '/autoload_real.php';

return ComposerAutoloaderInitccf1cf632ee945a32593ab9d031a56e5::getLoader();
```

5. ต่อมาใน ไฟล์ `vendor/composer/autoload_real.php` จะทำการเรียกไฟล์ `Classloader` มาทำการสร้างเป็นวัตถุชื่อ `$loader` แล้วก็

```php
spl_autoload_register(array('ComposerAutoloaderInit7ea998d8
 3f7753aae6cf282dc282bf3f', 'loadClassLoader'), true, true);
self::$loader = $loader = new \Composer\Autoload\ClassLoader();
spl_autoload_unregister(array('ComposerAutoloaderInit7ea998d
    83f7753aae6cf282dc282bf3f', 'loadClassLoader'));
```



6. เรียกไฟล์ `autoload_classmap` กับ `autoload_namespace`ขึ้นมาซึ่งในไฟล์ `autoload_classmap` จะเป็นรายชื่อของไฟล์ `package` ทั้งหมดที่อยู่ในโฟลเดอร์ `vendor` ส่วนในไฟล์ `autoload_namespace` จะเป็นรายชื่อของตัว `namespace` ที่ใช้เรียก `root path` ของไฟล์ `package` นั้นครับ

```php
$vendorDir = dirname(__DIR__);
$baseDir = dirname($vendorDir);

$map = require __DIR__ . '/autoload_namespaces.php';
foreach ($map as $namespace => $path) {
    $loader->set($namespace, $path);
}

$classMap = require __DIR__ . '/autoload_classmap.php';
if ($classMap) {
    $loader->addClassMap($classMap);
}

$loader->register(true);
```

7. พอเรียกเสร็จก็จะส่งกลับมาที่ ไฟล์ `autoload` แล้วก็ส่งค่าต่อไปให้ `ClassLoader` ของ `laravel`

```php
Illuminate\Support\ClassLoader::register();
```

8. ตามลงมาดูว่า `Classloader`ของ `laravel` มันทำยังไงต่อนะครับ

   ฟังก์ชัน `register`ทำการเรียกใช้ ฟังก์ชัน `load`

```php
public static function register()
{
        if ( ! static::$registered)
        {
spl_autoload_register(array('\Illuminate\Support\ClassLoader', 'load'));

                static::$registered = true;
        }
}
```

9. ฟังก์ชัน `load`จะทำการเรียกโฟลเดอร์ `package` ที่มีตามรายชื่อในตัวแปร `class`ที่ส่งมาตอนแรก

```php
public static function Load($class)
{
	$class = static::normalizeClass($class);

	foreach (static::$directories as $directory)
	{
	if (file_exists($path = $directory.DIRECTORY_SEPARATOR.$class))
		{
			require_once $path;

			return true;
		}
	}
}
```

10. เท่านี้ก็เสร็จการโหลด `package` ในโฟลเดอร์ `vendor` ครับ อะแต่ยังไม่หมดนะครับ ยังเหลือ package ที่โฟลเดอร์ workbench เรียกใช้คลาส `Starter`เพื่อทำการลงทะเบียน `package` ในโฟลเดอร์ `workbench`

```php
if (is_dir($workbench = __DIR__.'/../workbench'))
{
	Illuminate\Workbench\Starter::start($workbench);
}
```

โฟลเดอร์ workbech คือส่วนที่เราใช้ในการพัฒนา package ในเครื่องเราซึ่งยังไม่เสร็จหรือว่าทำขึ้นใช้เอง เราก็จะมาเริ่มต้นจากตรงนี้

package ถือว่าเป็นส่วนสำคัญที่จะต้องรู้ก่อนใน laravel 4 นี้เลย การเขียน package เป็นจะทำให้เราเลือก php libraly ตัวไหนก็ได้บน packagist.org ซึ่งเป็นศูนย์รวมของ libraly ของ php ที่ใช้มาตรฐาน psr-0,psr-1,psr-2 มาลงเป็น

pakage ใช้เอง ซึ่งผมก็ได้ศึกษาแล้วลองทำไปแล้วหลายตัวครับ ไว้จบเรื่องนี้ แล้วจะมาขียนเรื่องการสร้าง package ต่อครับ

ปล.ตัวแปร Laravel_Start ตอนแรกเราสามารถนำไปใช้ ลบกับค่าเวลาปัจจุบันเพื่อหาเวลาที่เว็บใช้ จัดการคำร้องขอ

ตอนที่ 2 Application

1. ขั้นตอนนี้จะเป็นการเริ่มสร้างตัวออปเจคหลักของ เว็บกันแล้วนะครับ
   เริ่มจากไฟล์ `bootstrap/start.php` เราจะดำดิ่งลงไปดูในคลาส `Application`

   ```
   $app = new Illuminate\Foundation\Application;
   ```

   ตรงคลาสนี้เป็นตัวหลักที่คอยขับเคลื่อนเว็บไซต์ของเราเลยครับ เพราะ มีการเรียกใช้ คลาสเข้ามาจำนวนมาก ฟังก์ชันในนี้ ผมจะไม่พยายามไปดูให้ตาลาย ให้รู้ว่าเรา มีฟังก์ชันที่เราต้องเข้าใจ คือ `App::share,App::bind,App::instance,App::make` เราไปดูต่อว่าเริ่มมาตัว `class Application`ทำอะไรต่อ

2. เนื่องจาก ทำการเรียกตัวคลาสเข้ามาแล้ว จึงทำการสร้างวัตถุใหม่ได้ทันทีเลยครับ
   สิ่งที่ต้องสร้างก่อนสามอันดับแรกเลยก็คือ `Class Exception` ใช้ในการจัดการข้อผิดพลาด
   `Class Routing` จัดการๆวิ่งของคำร้องขอ
   `Class Event` จัดการเหตุการณ์ต่าง ที่เกิดขึ้น

```
public function __construct(Request $request = null)
{
        $this['request'] = $this->createRequest($request);

        $this->register(new ExceptionServiceProvider($this));

        $this->register(new RoutingServiceProvider($this));

        $this->register(new EventServiceProvider($this));
}
```

3. กลับมาที่ ไฟล์ `start.php` ครับ ขั้นต่อมา เราจะทำการตรวจสถานะการทำงานของระบบนะครับ ว่าอยู่ในโหมดไหน ตรงนี้มีประโยชน์มากในกรณีเมื่อเรากำลังพัฒนาอยู่ก็ตั้งให้เป็น `development` ส่วนจะไปกำหนดที่ไหนค่อยมาพูดกันอีกทีครับ

```
$env = $app->detectEnvironment(array(

        'local' => array('your-machine-name'),

));
```

4. ทำการโหลดไฟล์ `path.php` ซึ่งจะเป็น `class`ใช้ในการกำหนดเส้นทางการเข้าหาโฟลเดอร์ต่างๆ

```
$app->bindInstallPaths(require __DIR__.'/paths.php');
```

5. ทำการเรียกใช้งานตัว `laravel` ละครับตรงนี้จะเป็นการ เรียกไฟล์ `start` ของตัว `laravel` ขึ้นมา

```
$framework = $app['path.base'].'/vendor/laravel/framework/src';

require $framework.'/Illuminate/Foundation/start.php';
```

ตอนที่ 3 Bootstrap

1. laravel เริ่มการตรวจว่ามีการติดตั้ง `Mcrypt` ไหม

```
if ( ! extension_loaded('mcrypt'))
{
        die('Laravel requires the Mcrypt PHP extension.'.PHP_EOL);

        exit(1);
}
```

2. ต่อมาก็ตรวจสภาวะการตั้งค่าก่อนว่าเรากำลัง ตั้งให้อยู่ในสถานะการทอบสอบไหม

```
if (isset($unitTesting))
{
        $app['env'] = $env = $testEnvironment;
}
```

3. คลาส `Facade` จะทำการเคลียร์ ตัวแทนของมัน
   แล้วก็สร้างขึ้นใหม่

```
Facade::clearResolvedInstances();

Facade::setFacadeApplication($app);
```

4. ทำการโหลดข้อมูลการตั้งค่าจากตัวแปร `$env` ขึ้นมาใส่ให้คลาส `Config` ครับ แล้วสร้างตัวแทนให้คลาส `Config`

```
$config = new Config($app->getConfigLoader(), $env);

$app->instance('config', $config);
```

5. ทำการตั้งค่าวันเวลาโดยโหลดค่ามาจากตัวแปรในไฟล์ `app.php`

```
$config = $app['config']['app'];

date_default_timezone_set($config['timezone']);
```

6. โหลดค่าชื่อย่อจากที่ไฟล์ `app.php` มา

```
    'aliases' => array(
        'Former' => 'Former\Facades\Illuminate',
        'Basset' => 'Basset\Facade',
        'Notification' => 'Krucas\Notification\Facades\Not
        'Confide' => 'Zizaco\Confide\ConfideFacade',
        'Ardent' => 'LaravelBook\Ardent\Ardent',
        'Date' => 'Carbon\Carbon',
        'Theme' => 'Teepluss\Theme\Facades\Theme',
```

มาให้ฟังก์ชัน `getInstance` มาทำการลงทะเบียนไว้

```
AliasLoader::getInstance($config['aliases'])->register();
```

7. ทำการอนุญาตให้ใช้คำร้องขอชนิด `put` ใช้ทำการแก้ไขข้อมูล กับ `delete` ใช้ในการลบข้อมูล เพื่อใช้ตอนทำ `restful` ในกรณีที่ `firewall` ไม่อนุญาตุให้เมทอดสองอันนี้ผ่าน เราจึงจำเป็นต้องทำการเปลี่ยนส่วนหัวของเมทอด `post` ให้กลายเป็นสองเมทอดที่โดนบล็อก

```
Request::enableHttpMethodParameterOverride();
```

8. ทำการวมคลาสทุกตัวที่โหลดมาเข้าด้วยกัน `$config['providers']` คือรายชื่อคลาสที่อยู่ในไฟล์ `app.php`

```
$providers = $config['providers'];

$app->getProviderRepository()->load($app, $providers);
```

9. เรียกใช้ฟังก์ชัน `$app->boot()` เพื่อเริ่มต้นการทำงาน แล้วเรียกไฟล์ `global.php` ซึ่งเป็นไฟล์ที่เราสามารถใช้กำหนดค่าเริ่มต้นเองได้

```
$path = $app['path'].'/start/global.php';

if (file_exists($path)) require $path;
```

10. รองสุดท้ายดึงไฟล์ที่เราใช้กำหนดสภาวะการตั้งค่าขึ้นมา

```
$path = $app['path']."/start/{$env}.php";

if (file_exists($path)) require $path;
```

11. สุดท้ายในบทนี้แล้วครับเรียกไฟล์ `routes.php` จบตอนครับ

ตอนที่ 4 Run

ตอนสุดท้ายแล้วครับ เราใกล้ได้คำตอบกลับหรือ `Response`

1. ทำการเขียน `session`
2. ค้นหา `Route` ที่ส่งมาว่าตรงกับตัวที่กำหนดไว้ไหม
3. สั่งให้ `filter` เช่น `before`,`after` ทำงาน

4. สร้างคำตอบกลับหรือ `Response`
5. ส่งคำตอบกลับ
6. เรียกใช้ `after filter` อย่างเช่น `log`

จบแล้วครับกว่าจะได้คำตอบกลับหนึ่งครั้งยาวนานไหมครับ

ที่มา :: Kenny Mayer Per Your Request

การเริ่มต้นอย่างรวดเร็ว

> หมายเหตุ :: url ที่เราจะเรียกใช้งานในตอนเริ่มแรกคือ localhost หรือ 127.0.0.1 ตามด้วยชื่อโฟลเดอร์ของเว็บที่เราสร้าง แล้วตามด้วย public นะครับ ยกตัวอย่าง 127.0.0.1/taqmaninw/public นะครับ

บทนี้จะทำให้เราเห็นภาพรวมของ laravel นะครับ

การติดตั้ง

ถ้าไม่รู้จัก composer แนะนำไปอ่าน ที่นี้ ก่อน เริ่มจากใช้คำสั่ง

```
composer create-project laravel/laravel [illegible] --prefer-dist
```

หลังจากนั้น composer จะทำการดาวน์โหลดไฟล์ต่างๆ มาเก็บที่โฟลเดอร์ที่เรากำหนดชื่อไว้

หลังจากนั้นก็ทำความรู้จักกับ โครงสร้างโฟลเดอร์ของ laravel เริ่มแรกเราต้องไปกำหนดค่าต่างๆ ที่โฟลเดอร์ `app/config` ก่อนในนี้ก็จะมีการตั้งค่าให้มากมายแต่เราอาจต้องการแค่ การกำหนดค่าเบื้องต้น

Routing

เราต้องกำหนด url ที่เราจะอนุญาตให้เข้าถึง ฟังก์ชันต่างก่อนที่ `app/routes.php` ตัวอย่างการสร้าง Route เบื้องต้น

```
Route::get('users', function()
{
    return 'Users!';
});
```

ตอนนี้เมื่อเราพิมพ์ชื่อโปรเจคของเราบนบราวเซอร์แล้วตามด้วย `/users` เราจะเห็นคำว่า `Users!` แสดงอยู่

การสร้าง Route ไปหา Controller

```
Route::get('users', 'UserController@getIndex');
```

ตอนนี้ `/user` จะถูกส่งไปที่ฟังก์ชน getIndex ของ UserController

การสร้าง View

ต่อมาเรามาสร้างไฟล์ที่จะใช้จัดการรูปแบบในการแสดงผลที่โฟลเดอร์ `app/views` เราสร้างไฟล์ที่ชื่อ `layout.blade.php` และ `users.blade.php`ต่อด้วย `layout.blade.php`

```
<html>
    <body>
        <h1>Laravel Quickstart</h1>

        @yield('content')
    </body>
</html>
```

ต่อมาในไฟล์ `users.blade.php` เราจะดึงไฟล์ layout มาลง

```
@extends('layout')

@section('content')
    Users!
@stop
```

เพื่อความไม่งงว่าตัวแปรเหล่านี้คืออะไร ตามไปดูที่นี่เลยครับ Blade documentation

ถ้าเราจะแสดงไฟล์ view ที่เราสร้างก็ต้องอาศัย Route ดังตัวอย่าง

```
Route::get('users', function()
{
    return View::make('users');
});
```

ต่อไปเราจะไปลุยดาต้าเบสกันนะครับ

การทำ Migration

เราจะใช้คลาส Migration ในการจัดการฐานข้อมูลนะครับ
เริ่มต้นด้วยการตั้งค่าในการเชื่อมต่อฐานข้อมูลก่อนที่ไฟล์ `app/database` ในค่าเริ่มต้นแล้วอาเรย์ `driver`จะเป็น `mysql` แล้วเราก็เปลี่ยนค่าตรง `mysql` เป็นข้อมูลในการเชื่อมต่อฐานข้อมูลของเรา

ต่อมาในการสร้าง migration เราใช้ คำสั่ง artisan ใน commandline จากในโฟลเดอร์โปรเจคของเรา ตัวอย่าง

```
php artisan migrate:make create_users_table
```

ต่อมาไฟล์ migrration จะไปโผล่ที่โฟลเดอร์ `app/database/migrations` ในไฟล์จะมีฟังก์ชัร `up` และ `down` เราจะสร้าง Schema เพื่อการจัดการฐานข้อมูล

ตัวอย่างการสร้าง Schema

```
public function up()
{
    Schema::create('users', function($table)
    {
        $table->increments('id');
        $table->string('email')->unique();
        $table->string('name');
        $table->timestamps();
    });
}

public function down()
{
    Schema::drop('users');
}
```

ต่อมาเราก็รันคำสั่ง

```
php artisan migrate
```

ถ้าอยากย้อนคำสั่ง `migrate` เราต้องใช้คำสั่ง `migrate:rollback`

Eloquent ORM

Eloquent คือ ชุดคำสั่งที่เราใช้ในการทำ sql query นั่นเองครับ ช่วยให้เราสะดวกสะบาย ทำงานได้รวดเร็วขึ้น เริ่มแรกเราต้องไปสร้าง `model` ที่โฟลเดอร์ `app/models` โดยตั้งชื่อว่า `User.php` ตัวอย่างการประกาศคลาสในโมเดล

```
class User extends Eloquent {}
```

ตัวอย่างการเรียกใช้ Eloquent Model ครับ

```
Route::get('users', function()
{
    $users = User::all();

    return View::make('users')->with('users', $users);
});
```

เมทอด `all`ที่ต่อจากเนมสเปซ `User` จะคิวรี่ค่าทั้งหมดจากตาราง `users` ส่วนใน `View` เราใช้ฟังก์ชัน `with` เพื่อดึงเฉพาะคอลัมน์ user ครับ

Displaying Data

ตัวอย่างการแสดงค่าที่มาจากฐานข้อมูลบนไฟล์ view ครับ

```
@extends('layout')

@section('content')
    @foreach($users as $user)
        <p>{{ $user->name }}</p>
    @endforeach
@stop
```

ต่อไปเราต้องไปเรียนรู้เรื่อง Eloquent และ Blade. หรือแวะเข้าไปอ่านเล่นๆ ก่อนที่เรื่องQueues และ Unit Testing.ถ้าต้องการใช้งานในระดับสูงต่อก็ไปที่ IoC Container. Happy In Laravel krub :)

Artisan CLI

Introduction

`Artisan` คือ ชุดคำสั่งที่ใช้เรียกงานผ่านทาง `command line`เพื่อช่วยให้จัดการงานต่างให้ง่าย รวดเร็วขึ้น ซึ่ง `laravel` นำ `Class console` ของ `symfony` มาปรับใช้

การใช้งาน

ในการเรียกดูคำสั่งทั้งหมดใช้คำสั่ง `list`

```
php artisan list
```

ทุกคำสั่งจะมีวิธีการใช้ให้เราดู โดยเพิ่ม parameter `help` เข้าไปนะครับ

ตัวอย่างการใช้งาน help

```
php artisan help migrate
```

เราสามารถเรียก พร้อมกับเปลี่ยนการตั้งค่าโดยรวมของเว็บโดยเพิ่มพารามิเตอร์ `--env`

ตัวอย่างนี้เราเรียกในการตั้งค่าแบบ local

```
php artisan migrate --env=local
```

จะเรียกดูรุ่นของ laravel ก็ได้โดยพารามิเตอร์ `--version`

```
php artisan --version
```

การสร้างคำสั่งขึ้นใช้งานเอง

เราสามารถสร้างคำสั่ง `artisan` ขึ้นมาใช้ โดยไฟล์จะเก็บที่โฟลเดอร์ `app/commands`
ถ้าเราไม่อยากเก็บไว้ตรงนี้ก็ไปตั้งค่าที่ไฟล์ `composer.json` ได้

การสร้างคำสั่ง

เริ่มสร้างคลาส

เราจะใช้คำสั่ง `command:make` ใน `command line` เพื่อสร้าง `class` ขึ้นมาก่อนครับ

ตัวอย่างการใช้ command line สร้างคำสั่ง

```
php artisan command:make FooCommand
```

ถ้าเราอยากจะเปลี่ยนที่อยู่ให้กับไฟล์คำสั่งของเรา ก็ใช้คำสั่งนี้ไปเลยครับ

```
php artisan command:make FooCommand --path="app/classes" --namespace="Classes"
```

การตั้งค่าคำสั่ง

เริ่มต้นโดยการตั้ง `name` และ `description` รวมถึงส่วนประกอบอื่นของคลาส, โดยค่าเหล่านี้จะไปปรากฏตอนคำสั่ง `artisan list` ฟังก์ชัน `fire` ใช้ในการเรียกฟังก์ชันต่างๆ ที่จะทำงานในคำสั่งนี้

การตั้งค่าต่างๆ

`getArguments` กับ `getOptions` เมทอด เป็นที่ๆ เราะทำการตั้งค่าจะต่างๆ ทั้ง พารามิเตอร์ที่ 1 ที่ 2
การตั้งค่าจะมีลักษณะการส่งค่าลงอาเรย์.

เมื่อเรา คำสั่งของเรามีการให้ป้อนพารามิเตอร์ ตัวของ array ต้องมีรูปแบบดังนี้

```
array($name, $mode, $description, $defaultValue)
```

ตัวแปร `mode` เรากำหนดให้เป็นแบบต้องมี `InputArgument::REQUIRED` หรือไม่มีก็ได้ `InputArgument::OPTIONAL`.

เมื่อเรากำหนดให้มีการใส่คำสั่งเพิ่มเติม ลักษณะอาเรย์จะเป็นแบบนี้

```
array($name, $shortcut, $mode, $description, $defaultValue)
```

ในการกำหนด `mode` ให้เป็นได้หลายๆแบบได้เช่น

`InputOption::VALUE_REQUIRED`, `InputOption::VALUE_OPTIONAL`, `InputOption::VALUE_IS_ARRAY`, `InputOption::VALUE_NONE`.

ตัวอย่างรูปแบบค่าที่ต้องป้อนให้เมื่อกำหนด mode เป็น `VALUE_IS_ARRAY`

```
php artisan foo --option=bar --option=baz
```

การเข้าถึงตัวแปร

เมื่อคำสั่งทำงาน เราก็ต้องมีตัวจัดการในการ ดึงค่าต่างๆ ในพารามิเตอร์ของการตั้งค่า ที่รับมา

การดึงค่าจากพารามิเตอร์เฉพาะตัว

```
$value = $this->argument('name');
```

การดึงค่าทั้งหมด

```
$arguments = $this->argument();
```

การรับค่าจากค่าการตั้งค่าแบบเฉพาะตัว

```
$value = $this->option('name');
```

การรับค่าจากค่าการตั้งค่าแบบทั้งหมด

```
$options = $this->option();
```

ส่งผลการทำงาน

มีประเภทของคำสั่งที่จะแสดงออกทาง commandline 4 ประเภท คือ `info` , `comment` , `question` และ `error` ทั้ง 4 มีรูปแบบ unicode เป็น ANSI

ส่งข้อมูลของคำสั่งออกทางหน้าจอ

```
$this->info('Display this on the screen');
```

ส่งข้อความออกไปทางหน้าจอ

```
$this->error('Something went wrong!');
```

ให้ผู้ใช้งานเลือก

เราสามารถใช้การถามคำถาม และ ยืนยัน เพื่อความรวดเร็วในการใช้งาน

การถามคำถาม

```
$name = $this->ask('What is your name?');
```

การถามคำถามและค่าที่ป้อนมาเป็นรูปแบบ

```
$password = $this->secret('What is the password?');
```

ยืนยันการเลือก

```
if ($this->confirm('Do you wish to continue? [yes|no]'))
{
    //
}
```

เราสามารถกำหนดค่าเริ่มต้นของคำสั่ง `confirm` ให้เป็น`true` หรือ `false` ได้

```
$this->confirm($question, true);
```

การลงทะเบียนคำสั่ง

เมื่อการสร้างคำสั่งเสร็จสิ้น เราต้องนำไปลงทะเบียนที่ไฟล์ `app/start/artisan.php` โดยใช้คำสั่ง `Artisan::add` เพื่อลงทะเบียน

ตัวอย่างการใช้งาน

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

ถ้าคำสั่งเราใช้ใน IoC container เราต้องใช้คำสั่ง `Artisan::resolve` เพื่อผูกคำสั่งของเราไปกับ IOC ด้วย

ตัวอย่างการใช้งาน

```
Artisan::resolve('binding.name');
```

การเรียกใช้คำสั่งอื่นร่วม

บางเวลาเราต้องการจะเรียกใช้คำสั่งอื่นๆ สามารถใช้ฟังก์ชัน `call` เรียกได้

ตัวอย่างการใช้งาน

```
$this->call('command.name', array('argument' => 'foo', '--option' => 'bar'));
```

Cache

การตั้งค่า

Laravel เตรียมรูปแบบของการ `cache` ไว้ให้เราใช้แล้วหลายตัวเลยนะครับ . ซึ่งรายชื่อและการตั้งค่าอยู่ที่ `app/config/cache.php`. ในไฟล์เราสามารถตั้งค่าให้กับ cache ที่เราชื่นชอบได้เลย Laravel สนับสนุนทั้ง Memcached กับ Redis อยู่แล้วครับ.

ก่อนจะทำการตั้งค่าใดๆ ลองไปศึกษาก่อนนะครับ ตัว laravel ใช้ `file` cache เป็นตัวเริ่มต้นนะครับ
ถ้าเว็บไซต์มีขนาดใหญ่ควรเปลี่ยนไปใช้ `Memcached` หรือ `APC` จะดีกว่า

การใช้งาน Cache

การเก็บค่าลง cache

```
Cache::put('key', 'value', $minutes);
```

เก็บลงในกรณีที่ไม่มีค่านั้นอยู่

```
Cache::add('key', 'value', $minutes);
```

ตรวจว่ามีค่าไหม

```
if (Cache::has('key'))
{
    //
}
```

ดึงค่าจากแคช

```
$value = Cache::get('key');
```

ดึงค่าโดยเรียกค่าเริ่มต้น

```
$value = Cache::get('key', 'default');

$value = Cache::get('key', function() { return 'default'; });
```

กำหนดให้ค่าชนิดนี้ไม่มีวันหมดอายุ

```
Cache::forever('key', 'value');
```

บางเวลาเราต้องการเรียกใช้ค่าจากแคช แต่ไม่มีค่าแล้ว เราสามารถเรียกใช้ `Cache::remember` โดยการดึงค่าจากฐานข้อมูลขึ้นมาเก็บไว้ในแคช

```
$value = Cache::remember('users', $minutes, function()
{
    return DB::table('users')->get();
});
```

และยังสามารถผสมคำสั่ง `remember` กับ `forever`

```
$value = Cache::rememberForever('users', function()
{
    return DB::table('users')->get();
});
```

เราสามารถเก็บค่าชนิดใดก็ได้เพราะรูปแบบการเก็บค่าใช้แบบ `serialize`

การลบค่าออกจากคีย์

```
Cache::forget('key');
```

การเพิ่มและการลดค่า

แคชทุกชนิดยกเว้น `file` กับ `database` สนับสนุนการทำเพิ่มและลดค่าของแคช

การเพิ่มค่า

```
Cache::increment('key');

Cache::increment('key', $amount);
```

การลดค่า

```
Cache::decrement('key');

Cache::decrement('key', $amount);
```

Cache Sections

> **หมายเหตุ:** Cache sections ไม่สนับสนุนแคชแบบ `file` หรือ `database`

Cache sections คือการให้เราสามารถจับกลุ่มให้กับแคชที่มีรูปแบบการเก็บค่าที่คล้ายๆ กันโดยใช้คำสั่ง`section`

ตัวอย่างการใช้งาน Cache section

```
Cache::section('people')->put('John', $john);

Cache::section('people')->put('Anne', $anne);
```

การเข้าถึงค่าของแคช

```
$anne = Cache::section('people')->get('Anne');
```

flush คือการลบค่าออก:

```
Cache::section('people')->flush();
```

การเก็บแคชไว้ในฐานข้อมูล

เมื่อเราจะใช้ฐานข้อมูลเก็บค่าแคช เราต้องเพิ่มตารางที่จะใช้เก็บก่อน
นี่คือตัวอย่างการสร้างตารางที่ใช้เก็บแคชในรูปแบบ ของ laravel

```
Schema::create('cache', function($table)
{
    $table->string('key')->unique();
    $table->text('value');
    $table->integer('expiration');
});
```

เราสามารถสร้างคำสั่ง artisan ขึ้นมาใช้โดยไฟล์จะเก็บที่โฟลเดอร์ `app/commands`
ถ้าเราไม่อยากเก็บไว้ตรงนี้ก็ไปตั้งค่าที่ไฟล์ `composer.json` ได้

การสร้างคำสั่ง

เริ่มสร้างคลาส

เราใช้คำสั่ง `command:make` ใน command line เพื่อสร้าง class ขึ้นมาก่อนครับ

ตัวอย่างการใช้ command line สร้างคำสั่ง

```
php artisan command:make FooCommand
```

ถ้าเราอยากจะเปลี่ยนที่อยู่ให้กับไฟล์คำสั่งของเรา ก็ใช้คำสั่งนี้ไปเลยครับ

```
php artisan command:make FooCommand --path="app/classes" --namespace="Classes"
```

เริ่มลงรายละเอียด

เริ่มต้นโดยการตั้ง `name` และ `description` รวมถึงส่วนประกอบอื่นของคลาส, โดยค่าเหล่านี้จะไปปรากฏตอนคำสั่ง `artisan list`. ฟังก์ชัน `fire` ใช้ในการเรียกฟังก์ชันต่างๆ ที่จะทำงานในคำสั่งนี้

การตั้งค่าต่างๆ

`getArguments` กับ `getOptions` เมทอด เป็นที่ๆ เราะทำการตั้งค่าจะต่างๆ ทั้ง พารามิเตอร์ที่ 1 ที่ 2 .
การตั้งค่าจะมีลักษณะการส่งค่าลงอาเรย์.

เมื่อเรา คำสั่งของเรามีการให้ป้อนพารามิเตอร์ array ต้องมีรูปแบบดังนี้

```
array($name, $mode, $description, $defaultValue)
```

ตัวแปร mode เรากำหนดให้เป็นแบบต้องมี `InputArgument::REQUIRED` ไม่มีก็ได้ `InputArgument::OPTIONAL`.

เมื่อเรากำหนดให้มีการใส่คำสั่งเพิ่มเติมลักษณะอาเรย์จะเป็นแบบนี้

```
array($name, $shortcut, $mode, $description, $defaultValue)
```

ในการกำหนด `mode` ให้เป็นได้หลายๆแบบได้เช่น: `InputOption::VALUE_REQUIRED`, `InputOption::VALUE_OPTIONAL`, `InputOption::VALUE_IS_ARRAY`, `InputOption::VALUE_NONE`.

ตัวอย่างรูปแบบค่าที่ต้องป้อนให้เมื่อกำหนด mode เป็น `VALUE_IS_ARRAY`

```
php artisan foo --option=bar --option=baz
```

การเข้าถึงตัวแปร

เมื่อคำสั่งทำงาน เราก็ต้องมีตัวจัดการๆ ดึงค่าต่างๆ ในพารามิเตอร์ การตั้งค่า ที่รับมา

การดึงค่าจากพารามิเตอร์เฉพาะตัว

```
$value = $this->argument('name');
```

การดึงค่าทั้งหมด

```
$arguments = $this->argument();
```

การรับค่าแบบเฉพาะตัว

```
$value = $this->option('name');
```

การรับค่าแบบทั้งหมด

```
$options = $this->option();
```

ส่งผลการทำงาน

มีประเภทของคำสั่งที่จะแสดงออกทาง commandline 4 ประเภท คือ `info` , `comment` , `question` และ `error` ทั้ง 4 มีรูปบบ unicode เป็น ANSI

ส่งข้อมูลของคำสั่งออกทางหน้าจอ

```
$this->info('Display this on the screen');
```

ส่งเออเรอออกไปทางหน้าจอ

```
$this->error('Something went wrong!');
```

ให้ผู้ใช้งานเลือก

เราสามารถใช้การถามคำถาม และ ยืนยัน เพื่อความรวดเร็วในการใช้งาน

การถามคำถาม

```
$name = $this->ask('What is your name?');
```

**การถามคำถามและค่าที่ป้อนมาเป็นรูปแบบ ** **

```
$password = $this->secret('What is the password?');
```

Asking The User For Confirmation

```
if ($this->confirm('Do you wish to continue? [yes|no]'))
{
    //
}
```

เราสามารถกำหนดค่าเริ่มต้นให้คำสั่ง `confirm` ให้เป็น`true` หรือ `false` ได้

```
$this->confirm($question, true);
```

การลงทะเบียนคำสั่ง

เมื่อการสร้างคำสั่งเสร็จสิ้น เราต้องนำไปลงทะเบียนที่ไฟล์ `app/start/artisan.php` โดยใช้คำสั่ง `Artisan::add` เพื่อลงทะเบียน

ตัวอย่างการใช้งาน

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

ถ้าคำสั่งเราใช้ใน IoC container,เราต้องใช้คำสั่ง`Artisan::resolve` เพื่อมัดคำสั่งของเราไปกับ IOC ด้วย

ตัวอย่างการใช้งาน

```
Artisan::resolve('binding.name');
```

การเรียกใช้คำสั่งอื่นร่วม

บางเวลาเราต้องการจะเรียกใช้คำสั่งอื่นๆ สามารถใช้ฟังก์ชัน `call` เรียกได้

ตัวอย่างการใช้งาน

```
$this->call('command.name', array('argument' => 'foo', '--option' => 'bar'));
```

การดึงค่าของการตั้งค่าหลักมาใช้

ส่วนการตั้งค่าหลักๆ ของเว็บเราจะอยู่ที่โฟลเดอร์ `app/config` ในบทนี้เราจะมาดูว่า laravel เตรียมฟังก์ชันอะไรให้เราใช้ในการดึงค่าจากไฟล์ทั้งหลายในโฟลเดอร์ config ออกมาใช้ได้บ้าง.

laravel เตรียม class ที่ชื่อว่า `Config` ไว้ให้เราแล้วนะครับ

ยกตัวอย่างการดึงค่า timezone ออกมา

```
Config::get('app.timezone');
```

เราสามารถกำหนดค่าของตัวแปรนั้นใหม่ได้ กรณีที่รูปแบบไม่เป็นไปตามที่เราต้องการ:

```
$timezone = Config::get('app.timezone', 'UTC');
```

สังเกตุว่าถ้าเป็นการเข้าถึงค่าในอาเรย์ของไฟล์ laravel จะใช้เครื่องหมายดอท ในการเข้าถึงนะครับ

กำหนดค่าแบบไม่ต้องเข้าไปในไฟล์เลย

```
Config::set('database.default', 'sqlite');
```

การกำหนดค่าแบบนี้จะไม่ไปเขียนทับการตั้งค่าในไฟล์ app.php นะครับ จะเกิดผลเฉพาะตรงที่เราประกาศไว้เท่านั้น.

การกำหนดชุดรูปแบบของการตั้งค่าพื้นฐาน

ในการพัฒนาเว็บเรามักจะเปิด การตั้งค่าต่างๆเพื่อที่จะเอื้ออำนวยให้เราทราบข้อมูล ได้มากที่สุด แต่ในกรณีที่เว็บออนไลน์แล้วการแสดง การแสดงข้อมูลการทำงานผิดพลาด การลืมไปแล้วว่าเคยทิ้งคำสั่ง debug ไว้ตรงไหน

เริ่มต้นสร้างไฟล์ชุดการตั้งค่าในโฟลเดอร์ `config` ยกตัวอย่างชื่อ `local`.ยกตัวอย่างการตั้งค่าในไฟล์ สมมุติเราต้องการใช้แคชแบบ file ก็ทำแบบตัวอย่างเลยครับ

```
<?php

return array(

    'driver' => 'file',

);
```

> **Note:** testing เป็นชื่อที่ถูกกำหนด ไว้กับ laravel แล้วว่าถ้าอยู่ในชื่อนี้การตั้งค่าทั้งหมดจะอยู่ในการโหมด unit test ฉะนั้น เราอย่าไปตั้งทับมันเลยครับ

ส่วนการตั้งค่าที่เราไม่ได้ตั้งไว้ จะอ้างอิงกลับไปที่ไฟล์หลักนะครับ

ต่อมาเราต้องไปตั้งค่าให้ตัว laravel รู้ว่าขณะนี้อยู่ในโหมดไหน โดยเข้าไปตั้งค่าที่ `bootstrap/start.php` ตัวโฟลเดอร์จะอยู่ข้างหน้าสุดเลย. เข้าไปค้นหา `$app->detectEnvironment` ตัวฟังชันจะใช้ค้นหารูปแบบการตั้งค่าของเว็บเรา

```
<?php

$env = $app->detectEnvironment(array(

    'local' => array('your-machine-name'),

));
```

เราก็จะเปลี่ยนให้เป็นเหมือนตัวอย่าง

```
$env = $app->detectEnvironment(function()
{
    return $_SERVER['MY_LARAVEL_ENV'];
});
```

ตัวอย่างการเรียกใช้

```
$environment = App::environment();
```

การปรับปรุงเว็บไซต์

เมื่อเราต้องการปิดเว็บเพื่อทำการปรับปรุง เราจะกำหนดเมทอด `App::down` ไว้ที่ `app/start/global.php` ซึ่งจะทำให้ทุกคำร้องถูกพาไปที่หน้า ที่บอกว่าตอนนี้ เว้บกำลังอยู่ในสถานะปรับปรุง.

ต้องการทำให้รวดเร็วขึ้นก็ใช้ command line ก็ได้

```
php artisan down
```

up เป็นคำสั่งให้เว็บกลับไปอยู่ในสถานะออนไลน์อีกครั้ง

```
php artisan up
```

ถ้าต้องการเปลี่ยนหน้าที่ใช้ในการแสดงผลก็เข้าไปตั้งค่าที่ `app/start/global.php` ตัวอย่าง

```
App::down(function()
{
    return Response::view('maintenance', array(), 503);
});
```

Controllers

การทำงานเบื้องต้น

แทนที่เราจะวางฟังก์ชันไว้ที่ไฟล์ `routes.php` ควรเขียนลงไปที่ controller ดีกว่าครับ จะทำให้โค้ดมีระเบียบเรียบร้อยอ่านง่าย

เราจะสร้าง Controllers ไว้ที่โฟลเดอร์ `app/controllers` โฟลเดอร์นี้จะถูกกำหนดที่อยู่ไว้ในตัวแปร `classmap` ในไฟล์ `composer.json` โดยเริ่มต้น .

ตัวอย่างการเรียกใช้ controller:

```
class UserController extends BaseController {

    /**
     *
     */
    public function showProfile($id)
    {
        $user = User::find($id);

        return View::make('user.profile', array('user' => $user));
    }

}
```

ทุก controller จะสืบทอดคลาส `BaseController` จะเก็บไว้ที่โฟลเดอร์ `app/controllers`โดยคลาส `BaseController` ก็สืบทอดต่อมาจาก `Controller` ต่อไปคือตัวอย่างการลงทะเบียน controller ไว้ที่ route ครับ

```
Route::get('user/{id}', 'UserController@showProfile');
```

ตัวอย่างนี้เป็นการเรียกใช้งาน controller แบบใช้ namespace นะครับ

```
Route::get('foo', 'Namespace\FooController@method');
```

เราสามารถตั้งชื่อย่อให้เพื่อสะดวกต่อการใช้งาน ตามตัวอย่างข้างล่างเลยครับ

```
Route::get('foo', array('uses' => 'FooController@method','as' => 'name'));
```

จะสร้างลิ้งไปที่ controller ก็ใช้ฟังก์ชัน `URL::action`

```
$url = URL::action('FooController@method');
```

ใช้ฟังก์ชัน `currentRouteAction` เพื่อเรียกคำร้องขอที่ผ่านมา

```
$action = Route::currentRouteAction();
```

Controller Filters

คือการกรองค่า ที่ส่งเข้ามายังตัวเว็บซึ่ง laravel ก็มีรูปแบบการทำ filter ไว้ให้อยู่แล้วเหมือนในตัวอย่างคือ เมื่อเว็บเริ่มทำงาน ก็ตรวจว่าได้ลงชื่อเข้าใช้งานไหม แล้วค่าที่ส่งมาปลอดภัยไหม ส่งเครื่องหมายแปลกประหลาดมาไหม โดยทำการตรวจด้วย filter ชื่อ csrf แล้วถ้ามีการเรียก fooAction กับ barAction ก็ทำการเก็บ log ไว้ด้วย

```
class UserController extends BaseController {

    /**
     * [illegible] _construct [illegible]
     */
    public function __construct()
    {
        $this->beforeFilter('auth');

        $this->beforeFilter('csrf', array('on' => 'post'));

        $this->afterFilter('log', array('only' =>
                            array('fooAction', 'barAction')));
    }

}
```

เราสามารถจับกลุ่มให้ตัวกรองได้ เหมือนในตัวอย่างนะครับ

```
class UserController extends BaseController {

    /**
     * Instantiate a new UserController instance.
     */
    public function __construct()
    {
        $this->beforeFilter(function()
        {
            //[illegible]
        });
    }

}
```

RESTful Controllers

restful controller ของ laravel คือการกำหนดว่าเมื่อมีการส่งคำขอแบบนี้ไปที่ฟังก์ชันนี้ให้ตอบสนองแบบไหนนะครับ เป็นการกรองตามชนิดของคำร้อง ทั้ง `get,post,put,delete` การกรองระดับชนิดของคำร้องขอ ทำให้เราจัดการได้ง่ายขึ้น

การกำหนดค่า restful controller ใน route

```
Route::controller('users', 'UserController');
```

เมื่อเรากำหนด ดังตัวอย่างด้านบนแล้ว คลาส UserController ก็จะรับแค่ค่าที่เรากำหนด
โดยเราจะใส่ชนิดของคำร้องขอที่เราจะรับไว้ที่หน้าตัวฟังก์ชัน

```
class UserController extends BaseController {

    public function getIndex()
    {
        // [illegible] get [illegible]
    }

    public function postProfile()
```

```
    {
        // [illegible] post [illegible]
    }

}
```

ในกรณีที่ชื่อฟังชันเรามีหลายคำใช้เครื่องหมาย - เพื่อเรียกได้ดังตัวอย่างเราเรียกใช้ฟังก์ชัน adminprofile ซึ่งมีสองคำ `users/admin-profile`

```
public function getAdminProfile() {}
```

Resource Controllers

Resource controllers คือการลงทะเบียน restful controller ของเรากับ Route มีคำสั่งใน command line ที่ช่วยให้เราสร้าง restful controller ได้รวดเร็วขึ้น. ตัวอย่าง เราอยากจะสร้าง controller เพื่อจัดการภาพเราก็ใช้คำสั่ง `controller:make` ตามตัวอย่าง

```
php artisan controller:make PhotoController
```

แล้วก็ลงทะเบียนบอก route ว่า controller นี้เป็น restful

```
Route::resource('photo', 'PhotoController');
```

การใช้คำสั่งอัติโนมัติสร้างจะทำให้เราได้ restful controller แบบเต็มรูปแบบ ยังเป็นการสร้างตัวอย่างการใช้งาน restful ให้ด้วย

รูปแบบของ restful controller ที่คำสั่ง artisan จะสร้างให้

Verb	Path	Action	Route Name
GET	/resource	index	resource.index
GET	/resource/create	create	resource.create

POST	/resource	store	resource.store
GET	/resource/{id}	show	resource.show
GET	/resource/{id}/edit	edit	resource.edit
PUT/PATCH	/resource/{id}	update	resource.update
DELETE	/resource/{id}	destroy	resource.destroy

ถ้าเราไม่ต้องการสร้างทั้งหมดตามในตารางก็กำหนดเป็นรายตัวไปเลยครับ ดังตัวอย่าง

```
php artisan controller:make PhotoController --only=index,show

php artisan controller:make PhotoController --except=index
```

แล้วก็กำหนดค่าใน route ให้ใช้งานได้เฉพาะฟังก์ชันก็ได้

```
Route::resource('photo', 'PhotoController',
                array('only' => array('index', 'show')));
```

การแสดงผลเมื่อมีคำร้องที่ไม่ถูกต้อง

เราจะใช้ฟังก์ชัน `missingMethod`เพื่อจัดการคำร้องขอที่ไม่ตรงตามที่เรากำหนดไว้
จะส่งไปไหนจะแสดงอะไรก็กำหนดได้เลยครับ:

ตัวอย่าง

```
public function missingMethod($parameters)
{
    //
}
```

การจัดการฐานข้อมูลเบื้องต้น

การตั้งค่า

เราจะไปตั้งค่าที่ `app/config/database.php`. ซึ่งเราจะไปตั้งค่าการเชื่อมต่อข้างในนี้ครับ

laravel ในขณะนี้สนับสนุน: MySQL, Postgres, SQLite, and SQL Server.

การทำ Queries

เราใช้คลาส `DB` ในการเรียกใช้การคิวรี่นะครับ

ตัวอย่างการคิวรี่

```
$results = DB::select('select * from users where id = ?', array(1));
```

ฟังก์ชัน`select`จะส่งค่าเป็น `array`กลับมาเสมอ

ตัวอย่างการเพิ่มข้อมูล

```
DB::insert('insert into users (id, name) values (?, ?)', array(1, 'Dayle'));
```

ตัวอย่างการแก้ไขข้อมูล

```
DB::update('update users set votes = 100 where name = ?', array('John'));
```

ตัวอย่างการลบข้อมูล

```
DB::delete('delete from users');
```

> **หมายเหตุ:** คำสั่ง `update` กับ `delete` จะคืนค่าเป็นจำนวนของแถวที่ได้ทำการจัดการไป.

ตัวอย่างการใช้คำสั่งทั่วไป

```
DB::statement('drop table users');
```

เราสามารถกำหนดให้ฟังก์ชันนี้ทำงานเมื่อมีการคิวรี่เกิดขึ้นโดยใช้ฟังก์ชัน `DB::listen`

ตัวอย่างการใช้งาน

```
DB::listen(function($sql, $bindings, $time)
{
    //
});
```

Database Transactions

เมื่อเราจะทำการคิวรี้หลายๆ คำสั่งเราจะใช้ฟังก์ชัน `transaction` ในการควบคุม เหมือนในตัวอย่างข้างล่าง

```
DB::transaction(function()
{
    DB::table('users')->update(array('votes' => 1));

    DB::table('posts')->delete();
});
```

การเข้าถึงการเชื่อมต่อฐานข้อมูล

สมมุติเว็บของเราใช้ฐานข้อมูลหลายชนิด สามารถใช้ฟังก์ชัน `DB::connection` ในการเรียกฐานข้อมูลเฉพาะที่เราต้องการดังตัวอย่าง

```
$users = DB::connection('foo')->select(...);
```

แล้วก็สามารถเชื่อมต่อใหม่ด้วยคำสั่ง reconnect

```
DB::reconnect('foo');
```

การเก็บประวัติการคิวรี่

โดยค่าเริ่มต้น laravel จะเก็บประวัติไว้ในหน่วยความจำอยู่แล้ว ในกรณีที่เว็บของเรามีคำร้องขอจำนวนมาก การเก็บประวัติจะใช้หน่วยความจำมาก เราจะใช้ฟังก์ชัน `disableQueryLog` เพื่อทำการหยุดเก็บประวัติ ตัวอย่าง

```
DB::connection()->disableQueryLog();
```

Eloquent ORM

ทำความรู้จักก่อน

Eloquent ORM คือการที่เราจำลองตารางเป็นคลาสแล้ว เรียกใช้งานเป็นชื่อของตาราง
นั้นเลยทำให้เข้าใจการพัฒนารวดเร็วขึ้น เข้าใจง่ายขึ้น ที่เรียกว่า Eloquent เพราะตัวมันมีความเรียบง่าย

การใช้งานเบื้องต้น

เริ่มต้นด้วยการสร้าง moedel ไว้ที่โฟลเดอร์ `app/models`

ตัวอย่างการสร้าง model

```
class User extends Eloquent {}
```

ถ้าคลาสนี้จะไม่ใช้ตารางตามชื่อ model เราก็สามารถใช้ตัวแปร `table` ในการกำหนดชื่อตารางที่เราจะใช้
เหมือนในตัวอย่าง

```
class User extends Eloquent {

    protected $table = 'my_users';

}
```

> **หมายเหตุ:** Eloquent จะถือว่าคอลัมน์ `id` เป็นคีย์หลักเสมอ เราสามารถใช้ตัวแปร `primaryKey`
> เพื่อกำหนดคีย์หลักได้เอง และเช่นเดียวกัน เราสามารถใช้ตัวแปร `connection`เพื่อกำหนดฐานข้อมูลที่เราจะใช้ใน
> model นี้

ถ้าในตารางของเรามีคอลัมน์ที่ชื่อ `updated_at` กับ `created_at` จะถูกใช้ในการเก็บเวลาที่ข้อมูลในแถวนี้ถูกเพิ่มหรือแก้ไข.
ถ้าเราไม่ต้องการเพียงแค่ตั้งค่าตัวแปร `$timestamps` ให้เป็น `false`

การคิวรี่โดยใช้ eoloquent

ค้นหาข้อมูลทั้งหมดจากตาราง user

```
$users = User::all(); //
```

ค้นหาตามเงื่อนไข

ค้นหาโดยค่า id เท่ากับ 1

```
$user = User::find(1); //
```

แสดงค่าในคอลัมน์ออกมา

```
var_dump($user->name); //
```

> **Note:** ทุกคำสั่งที่ใช้ใน query builder สามารถใช้กับ eloquent ได้เช่นกัน

การคิวรี่แล้วส่งต่อ

บางเวลาเมื่อค้นหาแล้วไม่เจอเราต้องการให้เกิดหน้าแสดงข้อผิดพลาดขึ้นมา สามารถใช้ฟังก์ชัน `findOrFail` เหมือนในตัวอย่างเลยครับ ถ้าค้นไม่เจอเราจะส่งไปหน้า 404 ทันที

```
$model = User::findOrFail(1);

$model = User::where('votes', '>', 100)->firstOrFail();
```

อยากจะสร้างการแสดงข้อผิดพลาดโดยที่เรากำหนดเองก็สามารถทำตามตัวอย่างเลยครับ สมมุติเราจะสร้างฟังก์ชัน `ModelNotFoundException`เราก็เรียกตัวคลาสหลักเข้ามาก่อน

```
use Illuminate\Database\Eloquent\ModelNotFoundException;

App::error(function(ModelNotFoundException $e)
{
    return Response::make('Not Found', 404);
```

```
});
```

ตัวอย่างการคิวรี่แบบหลายเงื่อนไข

```
$users = User::where('votes', '>', 100)->take(10)->get();

foreach ($users as $user)
{
    var_dump($user->name);
}
```

Eloquent Aggregates

```
$count = User::where('votes', '>', 100)->count();
```

ถ้าเราอยากเขียนคำสั่งคิวรี่ ขึ้นมาใช้เองก็ต้องใช้ฟังก์ชัน `whereRaw`

```
$users = User::whereRaw('age > ? and votes = 100', array(25))->get();
```

การส่งค่าอาเรย์ลงฐานข้อมูล

เราสามารถส่ง ค่าจำนวนมากอย่างเช่น อาเรย์ ลงฐานข้อมูลได้ง่ายๆ แต่ต้องใช้ตัวแปร

`fillable` เพื่อกำหนดว่าคอลัมน์ไหนที่สามารถใส่อาเรย์ได้

`guarded` เพื่อกำหนดว่าคอลัมน์ไหนใส่อาเรย์ลงไปไม่ได้

ตัวอย่างการกำหนดค่า fillable

```
class User extends Eloquent {

    protected $fillable = array('first_name', 'last_name', 'email');

}
```

ตัวอย่างการตั้งค่าตัวแปร guard

```
class User extends Eloquent {

    protected $guarded = array('id', 'password');

}
```

ตัวอย่างคอลัมน์ `id` and `password` เราจะไม่อนุญาติให้ทำการใส่ค่าที่มาในรูปแบบอาเรย์ลงไป

การป้องกันไม่ให้ทำการเพิ่มข้อมูลเป็นอาเรย์

```
protected $guarded = array('*');
```

เพิ่ม, ลบ, แก้ไข

ตัวอย่างการแก้ไขข้อมูลแบบไม่ใช้ namespace

```
$user = new User;

$user->name = 'John';

$user->save();
```

> **หมายเหตุ:** โดยเริ่มต้น laravel จะทำการเพิ่มค่าคีย์หลักให้อัตโนมัติ ถ้าเราไม่ต้องการก็ตั้งค่าตัวแปร `incrementing`ใน model ให้เป็น `false`.

เราสามารถใช้คำสั่ง create เพื่อสร้างข้อมูลใหม่ได้ แต่ก่อนหน้านั้นต้องกำหนดตัวแปร `fillable` หรือ `guarded` ไม่งั้นเพิ่มไม่ได้ติด error

การสร้างข้อมูลใหม่

```
$user = User::create(array('name' => 'John'));
```

ตัวอย่างการแก้ไขข้อมูล

```
$user = User::find(1);

$user->email = 'john@foo.com';

$user->save();
```

บางเวลาเราต้องบันทึกค่าในตารางที่อ้างอิงกัน เราจะใช้คำสั่ง `push`

บันทึกค่าพร้อมกับบันทึกลงตารางที่มีการเชื่อมกันอยู่

```
$user->push();
```

ตัวอย่างการลบข้อมูล

```
$user = User::find(1);

$user->delete();
```

ลบโดยกำหนด id เป็นเงื่อนไข

```
User::destroy(1);

User::destroy(1, 2, 3);
```

ตัวอย่างการลบแบบมีเงื่อนไข:

```
$affectedRows = User::where('votes', '>', 100)->delete();
```

ถ้าเราต้องการแก้ไขเฉพาะคอลัมน์ที่ใช้บันทึกเวลา เราจะใช้คำสั่ง `touch`

ตัวอย่างการใช้งาน

```
$user->touch();
```

การกำหนดว่าข้อมูลนี้ถูกลบแล้ว

เราสร้างตัวแปร `$softdelete` เพื่อบอก model ว่าไม่ต้องลบจริง เหมือนกับเราเอาไปเก็บไว้ในถังขยะก่อน ยังไม่ได้เอาไปเผาทิ้งจริงๆ

```
class User extends Eloquent {

    protected $softDelete = true;

}
```

แล้วก็เพิ่มคอลัมน์ `deleted_at` ลงในตาราง เพื่อกำหนดว่าข้อมูลแถวนี้ถูกลบแล้วหรือยัง

เมื่อเราเรียกคำสั่ง `delete` กับ model นี้คอลัมน์ `deleted_at` จะถูกเพิ่มค่าให้เป็นวันเวลาที่เราลบ เมื่อเราค้นหาข้อมูลโดยใช้ model นี้ข้อมูลแถวที่เราทำการ
ลบจะไม่ถูกดึงขึ้นมา

ตัวอย่างการค้นหา โดยรวมแถวที่ถูกตั้งค่าว่าลบแล้ว

```
$users = User::withTrashed()->where('account_id', 1)->get();
```

ตัวอย่างการค้นหา โดยค้นหาเฉพาะแถวที่ถูกตั้งค่าว่าลบแล้ว

```
$users = User::onlyTrashed()->where('account_id', 1)->get();
```

ถ้าต้องการยกเลิกการลบ ใช้คำสั่ง `restore` ได้เลยครับ

```
$user->restore();
```

หรือจะเรียกคืนเฉพาะแถวก็ตามตัวอย่างนี้เลย

```
User::withTrashed()->where('account_id', 1)->restore();
```

ฟังก์ชัน `restore` สามารถใช้กับความสัมพันธ์ได้ด้วย

```
$user->posts()->restore();
```

ถ้าต้องการลบข้อมูลจริงๆ ก็ใช้คำสั่ง `forceDelete`

```
$user->forceDelete();
```

คำสัง `forceDelete` ก็สามารถใช้กับความสัมพันธ์ก็ได้

```
$user->posts()->forceDelete();
```

ฟังก์ชัน `trashed` ใช้ในการตรวจว่าโมเดลนี้มีการตั้งค่า softdelete ไว้ไหม

```
if ($user->trashed())
{
    //
}
```

Timestamps การบันทึกเวลา

โดยค่าเริ่มต้น laravel ใช้คอลัมน์ `created_at` และ `updated_at` ในตารางของเราโดยอัตโนมัติ.

ตัวอย่างการยกเลิกการเก็บเวลาในการจัดการข้อมูล

```
class User extends Eloquent {

    protected $table = 'users';

    public $timestamps = false;

}
```

ฟังก์ชัน `freshTimestamp`ใช้ในการกำหนดรูปแบบวันเวลาที่เราจะเก็บ

ตัวอย่างการใช้งาน

```
class User extends Eloquent {

    public function freshTimestamp()
    {
        return time();
    }

}
```

Query Scopes

เราใช้คำนำหน้าฟังก์ชันว่า `scope` เพื่อทำการสร้างฟังก์ชันที่ใช้คิวรี่แบบเฉพาะของเราเอง:

ตัวอย่างการใช้งาน scope

```
class User extends Eloquent {

    public function scopePopular($query)
    {
        return $query->where('votes', '>', 100);
    }

}
```

การใช้งานคิวรี่ที่มาจาการใช้คำสั่ง scope

```
$users = User::popular()->orderBy('created_at')->get();
```

ความสัมพันธ์

การจัดการความสัมพันธ์ตารางใน laravel มี 4 รูปแบบ

- 1 ต่อ 1
- 1 ต่อ กลุ่ม

- กลุ่ม ต่อ กลุ่ม
- ความสัมพันธ์แบบซับซ้อน

1 ต่อ 1

ตัวอย่างความสัมพันธ์แบบ 1 ต่อ 1 ผู้ใช้งานมีโทรศัพท์ได้แค่เครื่องเดียว

ตัวอย่างความสัมพันธ์แบบ 1 ต่อ 1

```
class User extends Eloquent {

    public function phone()
    {
        return $this->hasOne('Phone');
    }

}
```

เราใช้ฟังก์ชันเป็นตัวกำหนดตารางที่เราจะเชื่อมด้วย ตัวอย่างข้างล่างการค้นหาโทรศัพท์ของผู้ใช้งานที่มี id เท่ากับ 1

```
$phone = User::find(1)->phone;
```

ถ้าเขียนเป็น php ธรรมดาก็จะได้แบบนี้ครับ

```
select * from users where id = 1

select * from phones where user_id = 1
```

โดยค่าเริ่มต้นแล้ว Eloquent จะใช้คอลัมน์ `user_id` ในตาราง `Phone` เป็น คีย์เชื่อม ถ้าเราไม่เอา จะเอาชื่อที่เราตั้งเองก็ใช้ตัวแปร `hasOne` เป็นตัวแก้ดังตัวอย่าง

```
return $this->hasOne('Phone', 'custom_key');
```

ในการเชื่อมโมเดลสิ่งที่สำคัญคือการตั้งค่าความสัมพันธ์ให้ตรงกันใน `Phone` model เราก็จะใช้ฟังก์ชัน `belongsTo` ในการเชื่อมกลับไปยัง `User`

ตัวอย่างการเชื่อมกลับไปยัง User Model

```
class Phone extends Eloquent {

    public function user()
    {
        return $this->belongsTo('User');
    }

}
```

เหมือนกับข้างบนครับ เราไม่เอา `user_id` เป็นคีย์เชื่อมก็ต้องกำหนดด้วย ตามตัวอย่าง

```
class Phone extends Eloquent {

    public function user()
    {
        return $this->belongsTo('User', 'custom_key');
    }

}
```

1 ต่อ กลุ่ม

ความสัมพันธ์แบบ 1 ต่อ กลุ่ม มีตัวอย่างคือ 1 โพสมีได้หลาย ความคิดเห็น
ตัวอย่างการใช้ hasMany

```
class Post extends Eloquent {

    public function comments()
    {
        return $this->hasMany('Comment');
    }

}
```

ตัวอย่างการค้นหาตารางที่เชื่อมกันอยู่

```
$comments = Post::find(1)->comments;
```

ตัวอย่างการค้นหาแบบหลายเงื่อนไขครับ ดังตัวอย่าง เราจะค้นหาความคิดเห็นที่มี title ชื่อ foo โดยเอาค่าแรกที่เจอก่อน

```
$comments = Post::find(1)->comments()->where('title', '=', 'foo')->first();
```

อีกครั้ง อย่าลืมเชื่อมกลับไปยังตารางที่เชื่อมมานะครับ

ตัวอย่างอีกครั้ง

```
class Comment extends Eloquent {

    public function post()
    {
        return $this->belongsTo('Post');
    }

}
```

กลุ่ม ต่อ กลุ่ม

กลุ่มต่อกลุ่ม จะเป็นความสัมพันธ์ที่ยุ่งยากพอสมควรเลยครับ เรามีตัวอย่างคือ ผู้ใช้งานมีสิทธิการใช้งานได้หลายสิทธิ์ ทั้งเรียกดู,ลบ,แก้ไข,เพิ่ม แล้วแต่ละสิทธิ์ก็ถูกใช้ได้ในหลายผู้ใช้งาน เราต้องมี 3 ตาราง `users`, `roles`, กับ `role_user`. ตาราง `role_user`จะเก็บ `user_id` กับ `role_id` เพื่อบอกว่า ผู้ใช้งานคนนี้มีสิทธิทำอะไรได้บ้าง

laravel ใช้ฟังก์ชัน `belongsToMany` ในการเชื่อมความสัมพันธ์:

```
class User extends Eloquent {

    public function roles()
    {
        return $this->belongsToMany('Role');
    }

}
```

ตอนนี้เราสามารถตรวจได้แล้วว่าผู้ใช้งานหมายเลข 1 มีสิทธิทำอะไรได้บ้าง

```
$roles = User::find(1)->roles;
```

ถ้าเราต้องการใช้ชื่อตาราง ตามใจเราก็สามารถทำได้โดยการ ส่งพารามิเตอร์ไป ดังตัวอย่างครับ

```
return $this->belongsToMany('Role', 'userroles');
```

หรือจะเปลี่ยนไปจนถึงชื่อคอลัมน์เลยก็ได้ แต่ต้องส่งชื่อ คอลัมน์ที่เราตั้งเองไปบอก model ด้วย

```
return $this->belongsToMany('Role', 'userroles', 'user_id', 'foo_id');
```

อย่าลืมเชื่อมความสัมพันธ์กลับมาด้วยนะครับ

```
class Role extends Eloquent {

    public function users()
    {
        return $this->belongsToMany('User');
    }

}
```

ความสัมพันธ์ที่ยุ่งยากมากขึ้น

เพื่อช่วยให้เข้าใจได้ง่ายขึ้นจะมีตัวอย่างมาให้ดูกันครับ

ตัวอย่างโครงสร้างตาราง

```
staff
    id - integer
    name - string

orders
    id - integer
    price - integer

photos
```

```
    id - integer
    path - string
    imageable_id - integer
    imageable_type - string
```

มันพิเศษตรงที่คอลัมน์ `imageable_id` กับ `imageable_type` ของตาราง `photos` ที่เราจะใช้เก็บคีย์ที่ใช้เชื่อมตาราง photo เข้ากับตาราง staff หรือ order ใช้เก็บคีย์เชื่อมร่วมกันได้ โดยใช้คอลัมน์
เมื่อเรากำหนดคีย์เชื่อมและชื่อของตารางที่เชื่อมไป ORM จะทำการตรวจสอบโดยใช้คอลัมน์ที่ `imageable_type` ในการหาว่าคีย์นี้เป็นของตารางไหน โดยเราต้องตั้งชื่อฟังก์ชันว่า `imageable` เราต้องประกาศ model แบบนี้ครับ

```
class Photo extends Eloquent {

    public function imageable()
    {
        return $this->morphTo();
    }

}

class Staff extends Eloquent {

    public function photos()
    {
        return $this->morphMany('Photo', 'imageable');
    }

}

class Order extends Eloquent {

    public function photos()
    {
        return $this->morphMany('Photo', 'imageable');
    }

}
```

ตอนนี้เราสามารถใช้ id ของ staff หรือ order มาค้นหารูปภาพได้

ตัวอย่าง

```
$staff = Staff::find(1);

foreach ($staff->photos as $photo)
{
    //
}
```

ความพิเศษจริง ๆอยู่ที่เมื่อเราใช้ `Photo` model ในการค้นครับ

ตัวอย่าง

```
$photo = Photo::find(1);

$imageable = $photo->imageable;
```

ความสัมพันธ์ที่ชื่อ `imageable` บน model `Photo` จะส่งข้อมูลของทั้ง `Staff` และ `Order` หรือตารางใดตารางหนึ่ง ขึ้นอยู่กับค่าที่เราใช้ค้นหาจะไปตรงกับ model ไหน

การคิวรี่โดยใช้ความสัมพันธ์เป็นเงื่อนไข

เราสามารถจำกัดผลการค้นหาด้วยฟังก์ชัน `has`

ค้นหาโดยจำกัดเฉพาะความสัมพันธ์

```
$posts = Post::has('comments')->get(); // [illegible] post [illegible] comment
```

You may also specify an operator and a count:

```
$posts = Post::has('comments', '>=', 3)->get(); // [illegible] post [illegible] comment [illegible] 3
```

การค้นหาแบบยืดหยุ่น

Eloquent ทำให้เราสามารถค้นหาแบบต่อเนื่องโดย

พยายามให้เราจำกัดขอบเขตการค้นหาให้ได้ลึกลงที่สุดเพื่อให้ได้เฉพาะข้อมูลที่ต้องการจริงๆ และพลิกแพลงรูปแบบของฟังก์ชันได้มากมาย ดังตัวอย่าง

```
class Phone extends Eloquent {

    public function user()
    {
        return $this->belongsTo('User');
    }

}

$phone = Phone::find(1);
```

แทนที่เราจะทำแบบข้างบน ซึ่งจะทำให้เราได้ค่าที่ไม่ต้องการออกมามาก เราก็เปลี่ยนมาใช้แบบข้างล่าง เราสามารถเข้าถึง อีเมล์ ของผู้ใช้งาน คนแรกได้เลย

```
echo $phone->user()->first()->email;
```

หรือจะให้สั้นได้อีก ก็ทำตามนี้เลยครับ

```
echo $phone->user->email;
```

Eager Loading

Eager loading มีเพื่อแก้ปัญหาการคิวรี่แบบ N + 1 ตัวอย่างคือ, ผู้แต่งหนึ่งคนสามารถแต่งหนังสือได้หลายๆ เล่ม ความสัมพันธ์จะออกมาแบบนี้

```
class Book extends Eloquent {

    public function author()
    {
        return $this->belongsTo('Author');
    }

}
```

แล้วการคิวรี้ที่มีปัญหาก็ประมาณนี้

```
foreach (Book::all() as $book)
{
    echo $book->author->name;
}
```

1 คิวรี้ จะทำการดึงค่าหนังสือทั้งหมดจากตาราง, แล้วการคิวรี่ครั้งต่อไปก็จะทำเหมือนกัน. ถ้ามีหนังสือ 25 เล่ม,จะมีการคิวรี่ถึง 26 ครั้ง คือเราใช้ข้อมูลทั้งหมดของตารางหนังสือไปค้นหาผู้แต่ง 1 ผู้แต่งก็จะไปดึงหนังสือทั้งหมดของเขาออกมา

```
select * from books

select * from authors where id = ?
```

นึกถึงเรามีข้อมูลเริ่มต้น 1000 แถว ปัญหานี้ส่วนใหญ่จะเกิดขึ้นกับความสัมพันธ์แบบ hasMany เพราะเราต้องนำทั้งหมดไปค้นหาต่อแล้วแต่ละแถวจะได้ผลลัพท์ออกมาหลายๆ แถวจำนวนผลการค้นหาที่มหาศาลจะทำให้การคิวรี่ช้ามากแต่ ซึ่งถ้าข้อมูลตั้งต้นยังมากกว่านี้ลงไปแต่งคิวรี่เองโดยใช้ Fluent Query Builder แต่ถ้ายังจะใช้ Eloquent ก็ยัง
โชคดีที่ laravel มีฟังก์ชัน `with` ใช้ในการทำให้เร็วขึ้น

```
foreach (Book::with('author')->get() as $book)
{
    echo $book->author->name;
}
```

sql ที่เกิดขึ้นจะมีหน้าตาแบบนี้ครับ เริ่มจากค้นหาหนังสือทั้หมดก่อนแล้ว ค่อยเอา id ที่ได้ไปค้นต่อในตาราง authors เราเปลี่ยนไปใช้ in แทน

```
select * from books

select * from authors where id in (1, 2, 3, 4, 5, ...)
```

จะทำให้เว็บของเราโหลดเร็วขึ้นอย่างมากเลยครับ

ตัวอย่าง การใช้ eager loading ในกรณีตารางมีการเชื่อมกับอีกหลายตาราง

```
$books = Book::with('author', 'publisher')->get();
```

จะใช้การทำ eager load กับคอลัมน์อื่นได้

```
$books = Book::with('author.contacts')->get();
```

In the example above, the `author` relationship will be eager loaded, and the author's `contacts` relation will also be loaded.

Eager Load Constraints

บางเวลาเราต้องการเฉพาะบางคอลัม์จาการทำ eager loading แล้วใส่เงื่อนไขเข้าไปอีก สามารถกำหนดได้ดั้งนี้ครับ:

```
$users = User::with(array('posts' => function($query)
{
    $query->where('title', 'like', '%first%');
}))->get();
```

Lazy Eager Loading

เราสามารถใช้การทำ eager loading ไปยังตารางที่เชื่อมกันได้เหมือนในตัวอย่างครับ
เราเข้าไปค้นต่อไปในตาราง publisher ที่เชื่อมกับตาราง author อีก

```
$books = Book::all();

$books->load('author', 'publisher');
```

การบันทึกข้อมูลแบบมีความสัมพันธ์

สมมุตอเราจะเพิ่มความคิดเห็นลงบทความนี้แล้วเราก็ต้องนำ id

ของบทความที่เราโพสความคิดเห็นใส่ไปมาใส่ในความคิดเห็นด้วย

ตัวอย่างการเก็บข้อมูลที่ต้องมีความสัมพันธ์

```
$comment = new Comment(array('message' => 'A new comment.'));

$post = Post::find(1);

$comment = $post->comments()->save($comment);
```

ในตัวอย่างคอลัมน์`post_id` จะถูกใส่ค่าให้อัติโนมัติ

Associating Models (Belongs To)

เมื่อเราจะทำการแก้ไขข้อมูลที่มีความสัมพันธ์แบบ กลุ่มต่อหนึ่ง เราต้องใช้ฟังก์ชัน `associate` ในการเพิ่มค่าคีย์เชื่อมไปยังตารางที่มีความสัมพันธ์อยู่ด้วย

```
$account = Account::find(10);

$user->account()->associate($account);

$user->save();
```

การเพิ่มข้อมูลแบบกลุ่มต่อกลุ่ม (Many To Many)

ตัวอย่างคือเราจะทำการเพิ่มความสามารถให้ผู้เราต้อง laravel มีฟังก์ชัน `attach` มาให้ใช้แล้ว

การเพิ่มข้อมูลแบบกลุ่มต่อกลุ่ม

```
$user = User::find(1);

$user->roles()->attach(1);
```

ตัวอย่างข้างล่าง เราจะทำการเพิ่มข้อมูลไปยังตารางที่ใช้เชื่อม สามารถส่งเป็นอาเรย์ก็ได้:

```
$user->roles()->attach(1, array('expires' => $expires));
```

เมื่อเพิ่มแล้วก็ลบได้ ฟังก์ชัน `detach` ใช้ลบค่าในตารางที่ใช้เชื่อม

```
$user->roles()->detach(1);
```

เราสามารถใช้ฟังก์ชัน `sync`เมทอด เพื่อการเพิ่มค่าไปยังตารางที่เชื่อมอยู่ด้วยได้ ในขณะที่เพิ่มลงในตารางหลัก

ตัวอย่างการใช้ sync กับความสัมพันธ์แบบกลุ่มต่อกลุ่ม

```
$user->roles()->sync(array(1, 2, 3));
```

Aตัวอย่างการใช้ sync กับตารางกลาง

```
$user->roles()->sync(array(1 => array('expires' => true)));
```

การใช้เมทอด `save` เพื่อทำการเพิ่มข้อมูลลงตารางที่เชื่อมกันอยู่

```
$role = new Role(array('name' => 'Editor'));

User::find(1)->roles()->save($role);
```

ในตัวอย่างเราสร้าง `Role` model แล้วแนบไปกับ User model. แล้วยังสามารถแนบอาเรย์เข้าไปได้อีก

```
User::find(1)->roles()->save($role, array('expires' => $expires));
```

การแก้ไขคอลัมน์ที่เก็บเวลาในตารางที่เชื่อมด้วย

ตัวอย่าง เมื่อเราแก้ไขข้อมูลในตาราง Comment เราต้องการแก้ไขข้อมูลในตาราง Post ในแถวที่เชื่อมกันด้วย laravel เตรียมฟังก์ชัน `touch` มาให้แล้ว วิธีการใช้งานในตัวอย่างเลยครับ

```
class Comment extends Eloquent {

    protected $touches = array('post');

    public function post()
```

```
    {
        return $this->belongsTo('Post');
    }

}
```

ตอนนี้ถ้าเราทำการแก้ไขข้อมูลในตาราง `Comment`, คอลัมน์ `updated_at` ข้อมูลในตาราง `Post`ที่เชื่อมด้วยก็จะถูกแก้ไขด้วย

การจัดการตารางที่ใช้เชื่อม

laravel เตรียมฟังก์ชัน `pivot` มาให้เราใช้ในการจัดการข้อมูลของตารางที่ใช้เชื่อมตรงกลางระหว่างสองตาราง ดังตัวอย่างเลยครับ

```
$user = User::find(1);

foreach ($user->roles as $role)
{
    echo $role->pivot->created_at;
}
```

คลาส `Role` model จะดึงค่าออกมาจากตารางกลาง โดยใช้ฟังก์ชัน`pivot` โดยอัติโนมัติ

โดยค่าเริ่มต้นแล้วค่าที่ได้จาก ตารางที่เป็นตัวเชื่อมจะมีค่าเดียวที่ใช้อ้างอิงไปยัง อีกตารางคือ id เท่านั้น แต่ถ้าเราต้องการเพิ่ม ก็ต้องเพิ่มไปตอนที่กำหนดความสัมพันธ์แบบในตัวอย่าง

```
return $this->belongsToMany('Role')->withPivot('foo', 'bar');
```

ตอนนี้ตัวแปร `foo` กับ `bar` จะถูกใช้กับฟังก์ชัน `pivot` ในการจัดการตาราง `Role`

และถ้าเราต้องการ คอลัมน์ `created_at` กับ `updated_at` เพื่อใช้กำหนดเวลา laravel มีฟังก์ชัน `withTimestamps` ซึ่งเราต้องกำหนดตอนประกาศความสัมพันธ์ครับ

```
return $this->belongsToMany('Role')->withTimestamps();
```


ต่อมาถ้าเราต้องการลบข้อมูลในตารางในตารางกลาง เพื่อความถูกต้องของข้อมูลเราจะใช้ฟังก์ชัน `detach`นะครับ

ตัวอย่างการใช้งาน

```
User::find(1)->roles()->detach();
```

เราจะทำการแก้ไขไม่ให้ผู้ใช้งานหมายเลข 1 มีสิทธิในการทำอะไรเลย

Collections

ข้อมูลที่เป็นผลลัพท์ของการค้นหานั้นจะกลับออกมาเป็น อาเรย์ eloquent อำนวนความสะดวกให้เราโดยมีฟังกชัน `contains` ให้ในการตรวจสอบข้อมูล

ตรวจว่าในผลลัพท์ที่ได้มามีข้อมูลที่มีคีย์หลัก เป็น 2 ไหม

```
$roles = User::find(1)->roles;

if ($roles->contains(2))
{
    //
}
```

เพื่อความสบายของเรายิ่งขึ้นไปอีกเมื่อค้นเสร็จก็เอาเฉพาะค่า role แปลงเป็นอาเรย์ หรือ json เสร็จสรรพเลย

```
$roles = User::find(1)->roles->toArray();

$roles = User::find(1)->roles->toJson();
```

แทนที่จะใช้การทำ foreach แบบปกติเหมือนเดิม eloquent มีฟังก์ชัน each กับ filter มาให้

การใช้งาน each และ filter

```
$roles = $user->roles->each(function($role)
       {

       });

$roles = $user->roles->filter(function($role)
```

```
        {

        });
```

เพิ่มการ Callback

```
$roles = User::find(1)->roles;

$roles->each(function($role)
{
    //
});
```

เรียงลำดับค่าที่อยู่ในอาเรย์ด้วยฟังก์ชัน sortby

```
$roles = $roles->sortBy(function($role)
{
    return $role->created_at;
});
```

บางครั้งเราต้องการเปลียนแปลงค่าทั้งออปเจคเลย eloquent ก็มีฟังก์ชัน `newCollection` ให้ใช้ในการเขียนทับ

ตัวอย่างการใช้งาน

```
class User extends Eloquent {

    public function newCollection(array $models = array())
    {
        return new CustomCollection($models);
    }

}
```

Accessors & Mutators

ถ้ายังไม่เข้าใจว่าสองฟังก์ชันนี้มันคืออะไร ทำอะไรได้บ้าง แนะนำเข้าไปอ่านที่ผมสรุปไว้ก่อนใน บล็อก ครับ
บางเวลาเราต้องการ จัดรูปแบบข้อมูลให้อยู่ในรูปแบบที่เราต้องการ ก่อนจะบันทึกหรือดึงมาใช้ eloquent เตรียมฟังก์ชัน

`getFooAttribute` แต่การตั้งชื่อฟังก์ชันคำเริ่มต้นของคำที่เป็นชื่อของคอลัมน์ ต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่ ในกรณีที่มีเครื่องหมาย _ มาคั่น คำหลังจากนั้นก็ต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่ครับ

ตัวอย่าง

```
class User extends Eloquent {

    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }

}
```

ในตัวอย่างเราทำการสร้าง accessor ของคอลัมน์ `first_name` ทีนี้ค่าที่เราส่งเข้าฟังก์ชันนี้ก็จะถูกส่งไปเก็บถูกที่ละครับ

การสร้างฟังก์ชัน Mutator ก็คล้ายๆ กัน

ตัวอย่าง

```
class User extends Eloquent {

    public function setFirstNameAttribute($value)
    {
        $this->attributes['first_name'] = strtolower($value);
    }

}
```

Date Mutators

โดยค่าเริ่มต้นแล้ว Eloquent จะทำการตั้งค่าให้คอลัมน์ `created_at`, `updated_at`, และ `deleted_at` ตามค่าเบื้องต้นของ php.ini อยู่แล้วนะครับ

แต่ถ้าเราต้องการแก้ไขหรือเรียกใช้งานแบบไม่อยากเข้าไปยุ่งตรงๆ ก็สามารถใช้ฟังก์ชัน `getDates`

แบบในตัวอย่างเลยครับ

```
public function getDates()
{
    return array('created_at');
}
```

ข้างในเราสามารถจัดการก่อนเอาไปใช้ได้เลย และ laravel ก็มีคลาสจัดการ วันเวลาที่มีฟังก์ชันหลากหลายมากอย่าง `Carbon` มาให้ใช้ด้วย แต่ต้องไปประกาศชื่อย่อในไฟล์ app.php ก่อนนะครับ โดยค่าเริ่มต้นแล้วไม่มี

Model Events

Eloquent เตรียมฟังก์ชันที่คอยดักจับเหตุการณ์ต่างๆ มาให้เราดังนี้ครับ `creating`, `created`, `updating`, `updated`, `saving`, `saved`, `deleting`, `deleted`. แต่ถ้าค่าที่ส่งกลับมาเป็น `false` เหตุการณ์ `creating`, `updating`, หรือ `saving` จะถูกยกเลิก

การยกเลิกการแก้ไขข้อมูล

```
User::creating(function($user)
{
    if ( ! $user->isValid()) return false;
});
```

การที่เราจะสร้างฟังก์ชันในการจัดการเหตุการณ์ของ model ต้องประกาศฟังชัน `boot` ก่อนนะครับ

การประกาศฟังก์ชัน boot

```
class User extends Eloquent {

    public static function boot()
    {
        parent::boot();

        // Setup event bindings...
    }

}
```

Model Observers

Eloquent มีคลาสชื่อ Observer ในการสร้างฟังก์ชันที่ใช้จัดการเหตุการณ์ ฟังก์ชัน`creating`, `updating`, `saving` ก็ตั้งตามเหตุการณ์ที่จะให้ฟังก์ชันนั้นจัดการครับ ตัวอย่าง

```
class UserObserver {

    public function saving($model)
    {
        //
    }

    public function saved($model)
    {
        //
    }

}
```

แล้วเราก็ต้องประกาศโดยใช้ฟังชัน `observe` แบบตัวอย่าง

```
User::observe(new UserObserver);
```

การแปลงค่าเป็น Arrays หรือ JSON

การแปลงผลลัพทที่ค้นมาให้กลายเป็น array

```
$user = User::with('roles')->first();

return $user->toArray();

return User::all()->toArray();
```

การแปลงผลลัพธ์ให้กลายเป็น json

```
return User::find(1)->toJson();
```

การใช้งาน Eloquent จากใน route เลย

```
Route::get('users', function()
{
    return User::all();
});
```

บางเวลาาเราไม่อยากให้บางคอลัมน์ถุกเรียกไปพร้อมกับ `toJson` หรือ `toArray` เราก็ใช้ตัวแปร `hidden` ในการนั้น

ตัวอย่างการใช้งาน

```
class User extends Eloquent {

    protected $hidden = array('password');

}
```

Errors & Logging

การแสดง Error

โดยค่าเริ่มต้นแล้วการแสดงข้อผิดพลาดจะถูกเปิดใช้งานอยู่แล้ว หากเราพัฒนาเว็บเสร็จก่อนจะส่งขึ้นโฮสติ้งก็ควรเข้าไปตั้งค่าตัวแปร `debug` ที่ `app/config/app.php` ให้เป็น `false`

การจัดการ Errors

โดยค่าเริ่มต้นแล้วการจัดการข้อผิดพลาดต่างๆเราจะทำในไฟล์ `app/start/global.php`

```
App::error(function(Exception $exception)
{
    Log::error($exception);
});
```

ถ้าเราอยากกำหนดการทำงานหลังจากเกิด `RuntimeException` เราก็ทำได้ดังตัวอย่างครับ

```
App::error(function(RuntimeException $exception)
{
    // Handle the exception...
});
```

เมื่อเราใช้การ return ในฟังก์ชันข้างล่างนี้ ค่าจะถูกส่งกลับไปยังบราวเซอร์เสมอครับ:

```
App::error(function(InvalidUserException $exception)
{
    Log::error($exception);

    return 'Sorry! Something is wrong with this account!';
});
```

การดักรอ PHP fatal errors เราใช้ฟังก์ชัน `App::fatal`

```
App::fatal(function($exception)
{
    //
});
```

HTTP Exceptions

กรณีข้อผิดพลาดที่เกิดขึ้นเพราะไม่พบหน้าที่เรียก (404), กับไม่มีสิทธิเข้าถึง (401) เราสามารถแก้ไขค่าที่จะไปแสดงได้ดังนี้

```
App::abort(404, 'Page not found');

App::abort(401, 'You are not authorized.');
```

ค่า 404 คือรหัสข้อผิดพลาดครับ

การจัดการข้อผิดพลาด 404

เราสามารถสร้างหน้าแสดงข้อผิดพลาดที่เรา สามารถออกแบบได้เอง แล้วต้องมาตั้งค่าแบบในตัวอย่างนี้ พารามิเตอร์ที่หนึ่งคือ ที่อยู่ของไฟล์ view

```
App::missing(function($exception)
{
    return Response::view('errors.missing', array(), 404);
});
```

การเก็บ log

คลาสที่ใช้ในการเก็บ log ของ laravel เป็นการใช้คลาส Monolog มาพัฒนาต่อ Laravel ตั้งค่าเริ่มต้นให้เก็บ log ทุกวันแล้วส่งไปเก็บไว้ใน `app/storage/logs`เราสามารถกำหนดค่า log ที่จะเขียนได้ดังตัวอย่าง

```
Log::info('This is some useful information.');

Log::warning('Something could be going wrong.');
```

```
Log::error('Something is really going wrong.');
```

ประเภทของ log ถูกประกาศไว้ในมาตรฐาน RFC 5424 มี **debug**, **info**, **notice**, **warning**, **error**, **critical**, และ **alert**.

ตัวอย่างการส่งค่าที่เป็นอาเรย์ให้ฟังก์ชัน info ที่อยู่ในคลาส log

```
Log::info('Log message', array('context' => 'Other helpful information'));
```

การดึงค่ามาใช้ก็ใช้ฟังก์ชันตามตัวอย่างเลยครับ

```
$monolog = Log::getMonolog();
```

เราสามารถกำหนดเหตุการณ์ที่เราจะให้เก็บ log ได้โดยใช้ฟังก์ชัน listen:

ตัวอย่าง

```
Log::listen(function($level, $message, $context)
{
    //
});
```

Events

การใช้งานเบื้องต้น

คลาส `Event` เตรียมมาให้เราใช้ในการดักจับเหตุการณ์ต่างๆที่เกิดขึ้นบน เว็บของเรา หมายเหตุ Event คือเหตุการใดที่เกิดขึ้นบนเว็บเรา ยกตัวอย่างการเพิ่มลบแก้ไข , Listener คือ ฟังก์ชันที่คอยดักจับเหตุการ , fire คือฟังก์ชันที่สั่งให้เกิดเหตุการขึ้นเพื่อให้ listerner ทำงาน

การดักรอฟังเหตุการณ์

```
Event::listen('user.login', function($user)
{
    $user->last_login = new DateTime;

    $user->save();
});
```

การสั่งให้เกิดเหตุการณ์ขึ้น

```
$event = Event::fire('user.login', array($user));
```

เราสามารถกำหนดลำดับเหตุการที่จะให้เกิดได้ โดยค่าลำดับที่เกิดเรียงจากน้อยไปหามาก.

การใช้ลำดับควบคุมการทำงาน

```
Event::listen('user.login', 'LoginHandler', 10);

Event::listen('user.login', 'OtherHandler', 5);
```

บางครั้งเราอยากจะให้ฟังก์ชันที่ดักฟังอยู่ทำงานแค่ครั้งเดียว เราสามารถใช้การ return `false` เพื่อหยุดฟังก์ชันนี้ได้

หยุดการทำงานของเหตุการ

```
Event::listen('user.login', function($event)
{
    // Handle the event...

    return false;
});
```

Wildcard Listeners

คือการดักฟังทุกเหตุการเลย ไม่เฉพาะเจาะจงแล้ว

การใช้งาน

```
Event::listen('foo.*', function($param, $event)
{
    // Handle the event...
});
```

ทีนี้ถ้ามีเหตุการณ์อะไรที่ขึ้นต้นด้วย `foo.`.ฟังก์ชันในตัวอย่างก็จะทำงาน

ใช้ Classes กับ Listeners

ในบางกรณี เราสามารถผูกคลาสเข้ากับเหตุการได้ โดย laravel จะใช้การทำ Laravel IoC container,ในการจัดการ

ตัวอย่าง

```
Event::listen('user.login', 'LoginHandler');
```

โดยค่าเริ่มต้นฟังก์ชัน `handle` ในคลาส `LoginHandler`จะถูกเรียกก่อนเลย เหมือนกับ _construct

ฟังก์ชัน handle จะถูกเรียกใช้เลย

```
class LoginHandler {

    public function handle($data)
    {
```

```
        //
    }

}
```

ถ้าเราไม่ต้องการ ก็ผูกเมทอดที่เราต้องการเข้าไปดังนี้ครับ

```
Event::listen('user.login', 'LoginHandler@onLogin');
```

การเรียงลำดับเหตุการ

ใช้ฟังก์ชัน `queue` กับ `flush` ในการเรียงลำดับการเกิดเหตุการ

การสร้างลำดับ

```
Event::queue('foo', array($user));
```

Event Subscribers

คือคลาสที่เราแบ่งกลุ่มฟังก์ชันที่ใช้จัดการเหตุการ โดยต้นทางมาจากฟังก์ชัน `subscribe`

ตัวอย่างการสร้างคลาส Subscriber

```
class UserEventHandler {

    /**
     * [illegible]
     */
    public function onUserLogin($event)
    {
        //
    }

    /**
     * [illegible]
     */
    public function onUserLogout($event)
```

```
    {
        //
    }

    /**
     * [illegible]
     *
     * @param  Illuminate\Events\Dispatcher  $events
     * @return array
     */
    public function subscribe($events)
    {
        $events->listen('user.login', 'UserEventHandler@onUserLogin');

        $events->listen('user.logout', 'UserEventHandler@onUserLogout');
    }

}
```

เมื่อประกาศคลาสแล้ว การนำไปใช้งานเราก็ต้องเอาไปลงทะเบียนกับคลาสหลัก

ตัวอย่าง

```
$subscriber = new UserEventHandler;

Event::subscribe($subscriber);
```

Facades

> **ความเห็นส่วนตัว** บทนี้จะต้องใช้จินตนาการเยอะหน่อยนะครับ เพราะเป็นเรื่องของแนวคิดที่ใช้สร้างจุดขายของตัว laravel เลยครับ ในการตัด $this-> ทิ้งไปทำให้โค้ดอ่านง่าย ทำความเข้าใจง่าย ลดบริมาณการเขียนลง ใครที่อยากจะเอาคลาสบน packagist มาทำเป็น package ใช้เองต้องอ่านครับ
>
> ## รู้จักด้านหน้าของตึกกันก่อน
>
> ถ้ายังไม่รู้จักว่ามันคืออะไรลองไปอ่านที่ผมสรุปไว้ก็ได้ครับ

Facades แปลเป็นไทยคือ ด้านหน้าของตึก ที่มาของมันคือการใช้ facade design pattern เข้ามาจัดการทำให้คลาสที่เราเรียกใช้งานกลายเป็นแบบ static แทนที่จะสร้างเป็นออปเจคเหมือนที่ผ่านมา การทำแบบ static คือการเรียกใช้คลาสนั้นตรงๆ เลยครับ การทำแบบนี้จะทำให้รูปแบบของฟังก์ชันดูเข้าใจได้ง่ายมาก บางครั้งเราไปเจอคลาสที่เจ๋งๆ และอยากเอาเข้ามาใช้ใน laravel เราก็อยากให้มันเรียกแบบ static ได้เพราะฉะนั้นเราจึงต้องมาดูบทที่นี้ครับ

> **Note:** ก่อนที่จะมาเจาะลึกลงไปในโครงสร้างหลักของ laravel แนะนำให้ไปอ่านบบทนี้ IoC container ก่อนครับ

หลักการเบื้องต้น

โดยโครงสร้างหลักของ laravel แล้ว, facade คลาสถูกวางให้ใช้ในการเรียกใช้งาน วัตถุที่ถูกลงทะเบียนไว้ในคลาส Container ในการสร้าง facade ขึ้นใช้เองนั้น เราต้องการแค่เมทอด `getFacadeAccessor`แค่ตัวเดียวครับ ตัวคลาส `Facade` หลักจะทำการใช้ `__callStatic()` ซึ่งเป็น magic-method ของ php จะไปทำการเรียกออปเจ็คของคลาสที่เราทำการลงทะเบียนไว้ที่คลาส `Service Provider` ของ ไลบราลี่ นั้นๆ

การประยุกต์ใช้งาน

ในตัวอย่าง, เรายกตัวอย่าง คลาส `Cache`. ซึ่งเรียกตัวฟังก์ชัน `get` ซึ่งตอนนี้เป็นรูปแบบ static นะครับ

```
$value = Cache::get('key');
```

แต่ถ้าเราตามเข้าไปดูตามเส้นทางนี้ `Illuminate\Support\Facades\Cache` เราจะไม่เห็นฟังก์ชัน `get` อยู่เลย

```
class Cache extends Facade {

    /**
     * Get the registered name of the component.
     *
     * @return string
     */
    protected static function getFacadeAccessor() { return 'cache'; }

}
```

การทำงานคือเมทอด `getFacadeAccessor()`จะทำการส่งค่าที่เราทำการผูกไว้ในคลาสหลัก โดยใช้สตริงที่เรากำหนดตรง return ในการค้นหาคลาสที่ตรงกันให้

เมื่อผู้ใช้งานอ้างอิงถึงตัว `Cache` คลาสในรูปแบบ static, Laravel จะผูก`cache` เข้ากับ IoC container และส่งคำขอฟังก์ชันตามในตัวอย่าง (กรณีนี้เป็น `get`)

ถ้าจะเขียน `Cache::get` แบบปกติจะได้แบบนี้ครับ

```
$value = $app->make('cache')->get('key');
```

การสร้าง Facades

การสร้าง Facade ให้ package หรือ คลาสภายนอกต้องการ 3 อย่างครับ

- การทำ IoC bind
- คลาส facade
- การสร้าง alias ให้ facade

ตัวอย่างการสร้าง package แบบง่ายครับ

`PaymentGateway\Payment`.

```
namespace PaymentGateway;

class Payment {

    public function process()
    {
        //
    }

}
```

เราต้องทำการผูกคลาสเข้ากับตัวคลาสหลักก่อน

```
App::bind('payment', function()
{
    return new \PaymentGateway\Payment;
});
```

ที่ๆ เราจะนำฟังก์ชันข้างบนไปเขียนไว้คือที่ไฟล์ service provider ที่ชื่อ `PaymentServiceProvider` โดยใส่ไว้ข้างในเมทอด `register`เราต้องเอาเส้นทางที่อยู่ของคลาส `PaymentServiceProvider` ไปใส่ตรงที่ `app/config/app.php` ด้วย

ต่อมา เราก็สร้างคลาส Facade ให้กับคลาส Payment

```
use Illuminate\Support\Facades\Facade;

class Payment extends Facade {

    protected static function getFacadeAccessor() { return 'payment'; }

}
```

สุดท้าย เรานำที่อยู่ของไฟล์ Facade ไปใส่ที่อาเรย์ชื่อ `aliases` ใน `app/config/app.php` และต้องใช้คำสั่ง `php artisan dump-autoload` ก่อนนะครับไม่งั้น laravel จะไม่เจอคลาส แล้วสุดท้ายเราสามารถเรียกเมทอด `process` ของ

คลาส `Payment` ในรูปแบบ static ได้แล้ว

```
Payment::process();
```

Helper Functions

เป็นคลาสอเนกประสงค์ที่รวมการจัดการอาเรย์,สตริง,ยูอาร์แอล อื่นๆ

Arrays

array_add

ฟังก์ชัน `array_add` ใช้เพิ่ม key / value ลงในอาเรย์ถ้าไม่มี key นั้นอยู่

```
$array = array('foo' => 'bar');

$array = array_add($array, 'key', 'value');
```

array_divide

ฟังก์ชัน `array_divide` จะทำการแบ่งอาเรย์ที่ส่งเข้าไปออก เป็นสองก้อน ก้อนหนึ่งเป็น key ก้อนหนึ่งเป็น value

```
$array = array('foo' => 'bar');

list($keys, $values) = array_divide($array);
```

array_dot

ฟังก์ชัน `array_dot` จะทำการแผ่อาเรย์หลายมิติ ออกเป็นมิติเดียว
โดยเราจะใช้เครื่องหมายดอทในการเข้าถึงหลังจากใช้ฟังก์ชันแแล้ว

```
$array = array('foo' => array('bar' => 'baz'));

$array = array_dot($array);

// array('foo.bar' => 'baz');
```

array_except

ฟังก์ชัน `array_except` ใช้ลบค่า key หรือ value ออกจากอาเรย์

```
$array = array_except($array, array('keys', 'to', 'remove'));
```

array_fetch

ฟังก์ชัน `array_fetch` จะส่งอาเรย์มิติเดียวที่เรากำหนดได้ว่าจะเอาค่าเฉพาะ คีย์ชื่ออะไร

```
$array = array(array('name' => 'Taylor'), array('name' => 'Dayle'));

var_dump(array_fetch($array, 'name'));

// array('Taylor', 'Dayle');
```

array_first

ฟังก์ชัน `array_first` จะส่งค่าแรกของอาเรย์คืนมา

```
$array = array(100, 200, 300);

$value = array_first($array, function($key, $value)
{
    return $value >= 150;
});
```

array_flatten

ฟังก์ชัน `array_flatten` ทำการแตกอาเรย์หลายมิติลงมาเหลือมิติเดียว

```
$array = array('name' => 'Joe', 'languages' => array('PHP', 'Ruby'));

$array = array_flatten($array);

// array('Joe', 'PHP', 'Ruby');
```

array_forget

ฟังก์ชัน `array_forget` ใช้ลบค่าออกจากอาเรย์โดยกำหนดตำแหน่ง ด้วยใช้เครื่องหมายดอท $array = array('names' => array('joe' => array('programmer')));

```
$array = array_forget($array, 'names.joe');
```

array_get

ฟังก์ชัน `array_get` ใช้ดึงค่าออกจากอาเรย์โดยกำหนดตำแหน่ง ด้วยเครื่องหมายดอท

```
$array = array('names' => array('joe' => array('programmer')));

$value = array_get($array, 'names.joe');
```

array_only

ฟังก์ชัน `array_only` ใช้ดึงค่าจากเฉพาะ key ที่เรากำหนด ได้ค่าเดียว

```
$array = array('name' => 'Joe', 'age' => 27, 'votes' => 1);

$array = array_only($array, array('name', 'votes'));
```

array_pluck

ฟังก์ชัน `array_pluck` ใช้ดึงค่าตามคีย์ที่กำหนด ได้หลายๆค่า

```
$array = array(array('name' => 'Taylor'), array('name' => 'Dayle'));

$array = array_pluck($array, 'name');

// array('Taylor', 'Dayle');
```

array_pull

ฟังก์ชัน `array_pull` ใช้ดึงค่าออกมาพร้อมกับลบไปด้วย.

```
$array = array('name' => 'Taylor', 'age' => 27);
```

```
$name = array_pull($array, 'name');
```

array_set

ฟังก์ชัน `array_set` ใช้เพิ่มค่าลงอาเรย์โดยกำหนดที่อยู่โดยใช้เครื่องหมายดอท

```
$array = array('names' => array('programmer' => 'Joe'));

array_set($array, 'names.editor', 'Taylor');
```

array_sort

ฟังก์ชัน `array_sort` ใช้เรียงลำดับค่าในอาเรย์

```
$array = array(
    array('name' => 'Jill'),
    array('name' => 'Barry'),
);

$array = array_values(array_sort($array, function($value)
{
    return $value['name'];
}));
```

head

ใช้คืนค่าแรกของ อาเรย์ มีประโยช์มากในการทำการเรียกฟังก์ชันแบบต่อเนื่อง

```
$first = head($this->returnsArray('foo'));
```

last

ใช้คืนค่าสุดท้ายของ อาเรย์ มีประโยช์มากในการทำการเรียกฟังก์ชันแบบต่อเนื่อง

```
$last = last($this->returnsArray('foo'));
```

Paths

ตัวแปรที่ใช้เก็บค่าที่อยู่ของโฟลเดอร์

app_path

เก็บค่าที่อยู่ของโฟลเดอร์ `application`

base_path

เก็บค่าที่อยู่ของเว็บระดับ root เลย

public_path

เก็บค่าที่อยู่ของโฟลเดอร์ `public`

storage_path

เก็บค่าที่อยู่ของโฟลเดอร์ `application/storage`

Strings

คลาสนี้ ใช้จัดการตัวอักษร เช่นแปลงเป็นตัวใหญ่ รูปแบบไวยกรณ์

camel_case

แปลงคำให้ขึ้นต้นด้วยตัวใหญ่ซึ่งเรียกว่า `camelCase`.

```
$camel = camel_case('foo_bar');

// fooBar
```

class_basename

ใช้ดึงชื่อคลาสจาก namespace path.

```
$class = class_basename('Foo\Bar\Baz');

// Baz
```

e

เรียกใช้ฟังก์ชัน `htmlentites` เพื่อกรองค่า

```
$entities = e('<html>foo</html>');
```

ends_with

ใช้ตรวจว่าในประโยคจบด้วยคำที่กำหนดไหม

```
$value = ends_with('This is my name', 'name');
```

snake_case

แปลงคำให้ไปอยู่ในรูปแบบ `snake_case` คืออักษรขึ้นต้นคำเป็นตัวเล็กแล้วแบ่งคำด้วยเครื่องหมายอันเดอร์สกอร์

```
$snake = snake_case('fooBar');

// foo_bar
```

starts_with

ใช้ตรวจว่าในประโยคขึ้นต้นด้วยคำที่กำหนดไหม

```
$value = starts_with('This is my name', 'This');
```

str_contains

ใช้ตรวจว่าในประโยคมีคำที่กำหนดไหม

```
$value = str_contains('This is my name', 'my');
```

str_finish

เพิ่มตัวอักษรที่กำหนดลงไปท้ายคำ

```
$string = str_finish('this/string', '/');

// this/string/
```

str_is

ตรวจว่าค่าที่ป้อนเข้ามาตรงกับรูปแบบที่กำหนดไหม

```
$value = str_is('foo*', 'foobar');
```

str_plural

แปลงตัวอักษรจากเอกพจน์เป็นพหูพจ์ ปล.เติม s,es,ies

```
$plural = str_plural('car');
```

str_random

สุ่มตัวอักษรขึ้นมาโดยกำหนดความยาวตามค่าที่ป้อนเข้ามา

```
$string = str_random(40);
```

str_singular

แปลงตัวอักษรจากเอกพจน์เป็นเอกพจน์

```
$singular = str_singular('cars');
```

studly_case

แปลงคำให้ไปอยู่ในรูปแบบ `StudlyCase` คืออักษรขึ้นต้นคำเป็นตัวใหญ่ถ้ามีเครื่องหมายอันเดอร์สกอร์ก็ลบออก

```
$value = studly_case('foo_bar');

// FooBar
```

trans

ใช้แปลภาษาเหมือนกับใช้ `Lang::get`.

```
$value = trans('validation.required'):
```

trans_choice

แปลโดยเริ่มต้นจากคำที่กำหนดโดยนับต่อไปตามค่าจากตัวแปร $count เหมือนกับ `Lang::choice`.

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

URLs

เป็นฟังก์ชันที่ใช้จัดการ URL

action

สร้างลิ้งจาก controller

```
$url = action('HomeController@getIndex', $params);
```

asset

สร้างลิ้งจากไฟล์ที่อยู่ในโฟลเดอร์ asset

```
$url = asset('img/photo.jpg');
```

link_to

สร้างลิ้งโดยกำหนดค่าต่างๆเอง

```
echo link_to('foo/bar', $title, $attributes = array(), $secure = null);
```

link_to_asset

สร้างลิ้งจากไฟล์ที่อยู่ในโฟลเดอร์ asset

```
echo link_to_asset('foo/bar.zip', $title, $attributes = array(), $secure = null);
```

link_to_route

สร้างลิ้งโดยอ้างอิงจากค่า route

```
echo link_to_route('route.name', $title, $parameters = array(), $attributes = array());
```

link_to_action

สร้างลิ้งเข้าไปหาฟังก์ชันใน controller

```
echo link_to_action('HomeController@getIndex', $title, $parameters = array(), $attributes = array());
```

secure_asset

สร้างลิ้งจากไฟล์ที่อยู่ในโฟลเดอร์ asset โดยใช้ https

```
echo secure_asset('foo/bar.zip', $title, $attributes = array());
```

secure_url

สร้างลิ้งที่เป็น https

```
echo secure_url('foo/bar', $parameters = array());
```

url

สร้างลิ้งจากการกำหนดเอง

```
echo url('foo/bar', $parameters = array(), $secure = null);
```

Miscellaneous

ฟังก์ชันอเนกประสงค์

csrf_token

สร้างค่า hash ที่ใช้ป้องกันการโจมตีแบบ csrf

```
$token = csrf_token();
```

dd

ใช้ในการดึงค่าทั้งหมดในตัวแปรออกมาแสดง

```
dd($value);
```

value

ดึงค่าจากฟังก์ชันที่ไม่มีชื่อ

```
$value = value(function() { return 'bar'; });
```

with

ใช้ดึงค่ากลับมาเป็นวัตถุ

```
$value = with(new Foo)->doWork();
```

Forms & HTML

เป็นคลาสที่ใช้จัดการ Form กับ html

การเปิดฟอร์ม

ตัวอย่าง

```
{{ Form::open(array('url' => 'foo/bar')) }}
    //
{{ Form::close() }}
```

โดยค่าเริ่มต้นชนิดของคำร้องขอจะเป็น `POST` ถ้าจะเปลี่ยนก็แค่ใส่พารามิเตอร์ไปเหมือนในตัวอย่างครับ

```
echo Form::open(array('url' => 'foo/bar', 'method' => 'put'))
```

สามารถกำหนดเป้าหมายของไฟล์ที่จะส่งค่าไปได้หลายรูปแบบตามตัวอย่างเลยครับ

```
echo Form::open(array('route' => 'route.name'))

echo Form::open(array('action' => 'Controller@method'))
```

จะกำหนดพารามิเตอร์โดยเฉพาะเลยก็ตามตัวอย่างครับ:

```
echo Form::open(array('route' => array('route.name', $user->id)))

echo Form::open(array('action' => array('Controller@method', $user->id)))
```

ถ้าจะสร้างฟอร์มมาอัพโหลดไฟล์ก็ต้องตั้งค่าแบบตัวอย่างครับ `files`

```
echo Form::open(array('url' => 'foo/bar', 'files' => true))
```

การป้องกัน CSRF

Laravel เตรียมการป้องกันโดยสร้างค่า hash ขึ้นจาก session ของ user แล้วสร้าง hidden form ขึ้นมาใส่ไว้ เราเพียงแต่ใช้เมทอด `token`ประกาศไว้ก็เสร็จแล้วครับ

ตัวอย่าง

```
echo Form::token();
```

การป้องกัน csrf จากใน route

```
Route::post('profile', array('before' => 'csrf', function()
{
    //
}));
```

การดึงค่าจากตารางมาใส่ในฟอร์ม

laravel เตรียมเมทอด `Form::model` มาเพื่อการนั้นครับ

ตัวอย่าง

```
echo Form::model($user, array('route' => array('user.update', $user->id)))
```

ถ้าชื่อของคอลัมน์ตรงกับ ชื่อของฟอร์มค่าก็จะปรากฏมาโดยอัติโนมัติ อย่างเช่นฟอร์มชื่อ `email`,ตรงกับโมเดลชื่อ `email` แต่ค่าที่จะปรากฏบนฟอร์มไม่ได้มีแค่ค่าที่มาจากโมเดลอย่างเดียว มีจาก session ค่าที่มาจากการส่งพารามิเตอร์อีก ลำดับการแสดงค่าจึงตามข้างล่างนี้ครับ 1. ค่าจาก Session (ค่าเก่าที่เกิดจาการป้อน) 2. ค่าจากการส่งพารามิเตอรี 3. ค่าจากโมเดล

ซึ่งเมื่อ server ส่งค่าการตรวจสอบค่าที่ป้อนมาว่าผิดพลาด ค่าที่ส่งมาจากโมเดลก็จะตามกลับขึ้นมาด้วย

> **หมายเหตุ:** เมือใช้ `Form::model`อย่าลืม `Form::close`!

การใส่ป้ายชื่อ

การใส่ป้ายชื่อให้ฟอร์ม

```
echo Form::label('email', 'E-Mail Address');
```

การใส่คลาสให้ฟอร์ม

```
echo Form::label('email', 'E-Mail Address', array('class' => 'awesome'));
```

> **หมายเหตุ:** หลังจากใส่ค่า label ชื่อฟอร์ม ค่า id ก็จะตั้งตามค่า label โดยอัติโนมัติ.

Text, Text Area, Password & Hidden Fields

ตัวอย่างฟอร์มที่ใช้รับค่า

```
echo Form::text('username');
```

กำหนดค่าเริ่มต้นให้ฟอร์ม

```
echo Form::text('email', 'example@gmail.com');
```

> **หมายเหตุ:** *hidden* และ *textarea* ฟังก์ชันใช้งานเหมือน *text* เมทอด

สร้างฟอร์มรับรหัสผ่าน

```
echo Form::password('password');
```

สร้างฟอร์มชนิดอื่น

```
echo Form::email($name, $value = null, $attributes = array());
echo Form::file($name, $attributes = array());
```

Checkboxes and Radio Buttons

สร้างฟอร์มชนิดเลือกค่า

```
echo Form::checkbox('name', 'value');

echo Form::radio('name', 'value');
```

สร้างฟอร์มชนิดเลือกค่าโดยค่าเริ่มต้นคือเลือกไว้แล้ว

```
echo Form::checkbox('name', 'value', true);

echo Form::radio('name', 'value', true);
```

File Input

สร้างฟอร์มอัพโหลดไฟล์

```
echo Form::file('image');
```

Drop-Down Lists

สร้างฟอร์มให้เลือกค่าแบบดรอบดาวน์

```
echo Form::select('size', array('L' => 'Large', 'S' => 'Small'));
```

สร้างฟอร์มชนิดเลือกค่าแบบดรอบดาวน์โดยค่าเริ่มต้นคือเลือกไว้แล้ว

```
echo Form::select('size', array('L' => 'Large', 'S' => 'Small'), 'S');
```

แบ่งกลุ่มให้ตัวเลือก

```
echo Form::select('animal', array(
    'Cats' => array('leopard' => 'Leopard'),
    'Dogs' => array('spaniel' => 'Spaniel'),
));
```

Buttons

สร้างปุ่มส่งค่า

```
echo Form::submit('Click Me!');
```

> **หมายเหตุ:** ถ้าจะสร้างปุ่มธรรมดาก็ใช้ button

Custom Macros

macro คือชุดของ html ที่เราเขียนเตรียมไว้ สามารถนำเอาไปแทรกตามใจเราได้

การสร้าง Form Macro

```
Form::macro('myField', function()
{
    return '<input type="awesome">';
});
```

เรียก Form Macro มาใช้

```
echo Form::myField();
```

IoC Container

Introduction

> หมายเหตุ :: ถ้าใครยังไม่ค่อยเข้าใจว่าสองตัวนี้มันคืออะไรลองไปอ่านที่ผมสรุปไว้ที่นี่ก่อนครับ

IOC คือคลาสที่ใช้ในการจัดการ `library` ภายนอกที่เรานำเข้ามาใช้ หรือที่ดึงเข้ามาใช้โดย `composer`ต่อไปนี้จะเรียกย่อๆว่า IOC นะครับ
IOC จะช่วยในการเรียกใช้งานคลาสต่าง การทำความเข้าใจ IOC ถือว่าเป็นหัวใจ
เลยในการปูทางสู่การทำเว็บขนาดใหญ่สำหรับ
laravel นะครับ เพราะหลักการนี้จะเกี่ยวโยงไปถึงเรื่อง `Service Provider` กับ `Facade`

การใช้งานเบื้องต้น

การใช้งาน IOC ในการผูก object

```
App::bind('foo', function($app)
{
    return new FooBar; //
});
```

ง่ายคือต่อไปนี้พารามิเตอร์ชื่อ foo จะใช้เป็นตัวแทนของ class FooBar ที่ถูกสร้างเป็นวัตถุแล้ว

การเรียกใช้งาน

```
$value = App::make('foo');
```

เมื่อเราเรียกใช้งาน `App::make` เมทอด ฟังก์ชันข้างบนก็จะถูกเรียกใช้งาน ตัวแปร value
ก็จะรับคุณสมบัติต่างๆ ของคลาส FooBar เข้ามา

บางครั้งเราไม่ต้องการสร้าง `instane` ทุกครั้งที่รีเฟรช laravel มีฟังก์ชัน singleton มาให้ใช้ในการนี้เลยครับ

ตัวอย่างการใช้งาน

```
App::singleton('foo', function()
{
    return new FooBar;
});
```

ถ้าไม่ต้องการจัดเพิ่มเข้าไปทั้งคลาสจะยัดเข้าไปเป็นออปเจ็คก็ใช้ฟังก์ชัน `instance` ได้เลยครับ

ตัวอย่างการผูกออปเจคเข้าไป

```
$foo = new Foo;

App::instance('foo', $foo);
```

Automatic Resolution

ตัวอย่างการผูกคลาสอย่างรวดเร็วขึ้นโดยไม่ต้องใช้ `App::bind` แล้วแต่ขอให้ชื่อตรงกันก็พอ

ตัวอย่าง

```
class FooBar {

    public function __construct(Google $baz)
    {
        $this->baz = $baz;
    }

}

$fooBar = App::make('FooBar');
```

ในตัวอย่างนี้ ตัวแปร $foofBar จะเก็บค่าคลาส `Google` ที่ถูกแทรกเข้ามา

laravel จะทำการเรียกใช้ reflection class ของ php เพื่อทำการตรวจสอบค่าต่างๆ ในคลาสนั้นให้เองครับ

บางกรณีคลาสที่เราจะใช้งานดันไปดึงคลาสที่เป็น interface เข้ามาใช้ด้วย เพื่อการนั้นเราต้องใช้ `App::bind` เมทอดในการผูก ดังตัวอย่าง

ตัวอย่างการผูก class ที่เป็นเรียกใช้งาน interface

```
App::bind('UserRepositoryInterface', 'DbUserRepository');
```

คลาสที่เราเรียกใช้งาน

```
class UserController extends BaseController {

    public function __construct(UserRepositoryInterface $users)
    {
        $this->users = $users;
    }

}
```

ตอนนี้ `UserRepositoryInterface` จะถูกเรียกใช้งานแล้ว

การประยุกต์ใช้งาน

เพื่อความยืดหยุ่นในการทดสอบและใช้งาน Laravel เตรียมการให้เราใช้งาน IOC ไในหลายกรณีเลยครับ

ตัวอย่างการเรียกใช้งาน Class OrderRepository

```
class OrderController extends BaseController {

    public function __construct(OrderRepository $orders)
    {
        $this->orders = $orders;
    }

    public function getIndex()
    {
```

```
        $all = $this->orders->all();

        return View::make('orders', compact('all'));
    }

}
```

ในตัวอย่างคลาส `OrderRepository` แทรกเข้าไปโดยอัติโนมัติเมื่อ คลาส `OrderController` ทำงาน เมื่อมีการทำ unit testing คลาส `OrderRepository` ก็จะถูกเพิ่มเข้ามาเหมือนกัน

Filters, composers, และ event handlers อยู่นอกเหนือการทำงานของ IoC container เมื่อเราจะใช้ต้องทำตามตัวอย่างครับ

การใช้งาน View::composer,Route::filter และ Event::listen กับ IOC

```
Route::filter('foo', 'FooFilter');

View::composer('foo', 'FooComposer');

Event::listen('foo', 'FooHandler');
```

Service Providers

Service providers เป็นการจับคลาส IOC ที่ทำงานคล้ายกันเข้ามาไว้ในที่เดียวกัน.
แล้วเรียกใช้งาน ด้วย facade
โดยหลักแล้วทุกคลาสหลักของ laravel ใช้การทำ service provider
ในการจัดการเราสามารถเข้าไปดูได้ตรงที่ตัวแปรอาเรย์ `providers` ตรงที่`app/config/app.php`

จะสร้าง Service Provider ขึ้นมาใช้กับคลาสของเราเริ่มแรกต้องดึงคลาส `Illuminate\Support\ServiceProvider` และประกาศเมทอด `register`

ตัวอย่างการสร้าง Service Provider

```
use Illuminate\Support\ServiceProvider;
```

```
class FooServiceProvider extends ServiceProvider {

    public function register()
    {
        $this->app->bind('foo', function()
        {
            return new Foo;
        });
    }

}
```

ในเมทอด `register` คลาส IOC จะเป็นตัวแปร `$this->app` ถ้าเราทำเสร็จแล้วก็ต้องเอาเส้นทางที่อยู่ของคลาส Provider ของเราไปเพิ่มในอาเรย์ `providers` ใน `app.php` ด้วย

การเรียกใช้งาน Service Provider ด้วยเมทอด `App::register`

ตัวอย่าง

```
App::register('FooServiceProvider');
```

Container Events

ใน IOC ก็มี event อยู่ชื่อเมทอดว่า `resolving`

การใช้งาน resoliving เพื่อรอดูว่ามี IOC ตัวไหนทำงานบ้าง

```
App::resolving(function($object)
{
    //
});
```

Localization

แนะนำ

คลาส `Lang` จะทำหน้าที่แปลภาษาในเบื้องต้นให้กับเมนูหรือป้ายกำหับต่างๆ ในเว็บของเรา

ไฟล์ที่เก็บข้อมูลภาษา

ถูกเก็บไว้ที่โฟลเดอร์ `app/lang` โดยโครงสร้างจะเป็นแบบนี้

```
/app
    /lang
        /en
            messages.php
        /es
            messages.php
```

ไฟล์ที่เก็บภาษาจะเก็บในรูปแบบอาเรย์:

ตัวอย่างของไฟล์ภาษา

```
<?php

return array(
    'welcome' => 'Welcome to our application'
);
```

โดยค่าเริ่มต้นแล้ว ค่าภาษาจะถูกกำหนดไว้ที่ `app/config/app.php` แต่ถ้าจะตั้งเราจะตั้งแบบไม่ให้คลุมไปทั้งเว็บก็ใช้เมทอด `App::setLocale`

ตัวอย่าง

```
App::setLocale('es');
```

การใช้งานเบื้องต้น

การดึงค่าจากไฟล์ภาษา

```
echo Lang::get('messages.welcome');
```

ฟังก์ชัน `get` ใช้ดึงค่าโดยมีพารามิเตอร์คือชื่อภาษาและแถวที่ต้องการดึง

การส่งพารามิเตอร์ไป

พารามิเตอร์ที่สองจะใช้รับค่าที่จะส่งมา

```
'welcome' => 'Welcome, :name',
```

ตัวอย่างการใช้งาน

```
echo Lang::get('messages.welcome', array('name' => 'Dayle'));
```

ตรวจว่าในไฟล์ภาษามีคอลัมน์นี้อยู่

```
if (Lang::has('messages.welcome'))
{
    //
}
```

Pluralization

Pluralization คือโครงสร้างไวยากรณ์ของแต่ละภาษาที่มีความแตกต่างกัน แต่เราใช้เครื่องหมายในการสร้างตัวเลือกระหว่างเอกพจน์กับพหูพจน์:

```
'apples' => 'There is one apple|There are many apples',
```

แล้วเราก็ใช้ฟังก์ชัน `Lang::choice` ในการเลือก

```
echo Lang::choice('messages.apples', 10);
```

เนื่องจากการแปลภาษาของ laravel สืบทอดมาจากของ Symfony เราจึงสามารถสร้างเงื่อนไขที่ซับซ้อนดังตัวอย่างได้

```
'apples' => '{0} There are none|[1,19] There are some|[20,Inf] There are many',
```

Mail

การตั้งค่า

Laravel นำไลบราลี่ SwiftMailer มาใช้งาน การตั้งค่าอยู่ที่ `app/config/mail.php`,โดยจะให้เราเปลี่ยน SMTP host, port, และ username กับ password, แล้วก็ค่า `from` คือค่าเริ่มต้นของชื่อผู้รับ. ถ้าเราต้องการใช้ไลบรารี `php mail` ในการส่งก็เพียงเปลี่ยน `driver` เป็น `mail`

การใช้งานเบื้องต้น

ฟังก์ชัน `Mail::send` ใช้ในการส่งอีเมล์

```
Mail::send('emails.welcome', $data, function($message)
{
    $message->to('foo@example.com', 'John Smith')->subject('Welcome!');
});
```

เมทอด `send` ตัวแปรแรกคือไฟล์ html ที่เป็นรูปแบบข้อมความในเมล์. ตัวที่สองคือข้อมูลที่จะเขียนลงเมล์ `$data` ซึ่งจะถูกส่งไปยัง view ตัวที่สามเป็นฟังก์ชันที่ใช้กำหนดค่าต่างๆของอีเมล์

> **Note:** ตัวแปร `$message` คือออปเจ็คของตัว Swiftmailer class ซึ่งเราจะใช้กำหนดค่าต่างๆของเมล์

```
Mail::send(array('html.view', 'text.view'), $data, $callback);
```

ตัวอย่างคือเราเลือกที่จะส่งไปในรูปแบบใด `html` หรือ `text`

```
Mail::send(array('text' => 'view'), $data, $callback);
```

ตัวอย่างการปรับแต่งเนื้อหาภายในเมล์:

```
Mail::send('emails.welcome', $data, function($message)
{
```

```
    $message->from('us@example.com', 'Laravel');

    $message->to('foo@example.com')->cc('bar@example.com');

    $message->attach($pathToFile);
});
```

เมื่อจะทำการแนบไฟล์เราต้องใส่นามสกุลกับชื่อให้มันด้วย:

```
$message->attach($pathToFile, array('as' => $display, 'mime' => $mime));
```

การแทรกไฟล์ไว้ระหว่างบรรทัด

เราสามารถแนบรูปไปโดยไม่ให้แสดงได้โดยใช้ฟังก์ชัน `embed`

ตัวอย่างการใช้งาน

```
<body>
    Here is an image:

    <img src="<?php echo $message->embed($pathToFile); ?>">
</body>
```

เรียงลำดับการส่งอีเมล์

Laravel เตรียมคลาส Queue มาให้เราใช้ในการเรียงลำดับการส่งอีเมล์

ตัวอย่าง

```
Mail::queue('emails.welcome', $data, function($message)
{
    $message->to('foo@example.com', 'John Smith')->subject('Welcome!');
});
```

เราสามารถหน่วงเวลาการส่งโดยใช้ฟังก์ชัน `later` ตามตัวอย่างครับ

```
Mail::later(5, 'emails.welcome', $data, function($message)
{
    $message->to('foo@example.com', 'John Smith')->subject('Welcome!');
});
```

ถ้าเรามีหลายคิว มีฟังก์ชันให้เราเรียงคิวอีก คือ `queueOn` และ `laterOn`

```
Mail::queueOn('queue-name', 'emails.welcome', $data, function($message)
{
    $message->to('foo@example.com', 'John Smith')->subject('Welcome!');
});
```

Mail & Local Development

ในการพัฒนานั้น เรายังไม่ต้องใช้งานเมล์จริงๆในการส่งก็ได้ laravel เตรียมฟังก์ชัน `Mail::pretend` หรือตั้งค่า `pretend` ใน `app/config/mail.php` เป็น `true`. เพื่อเข้าสู่ `pretend` mode ข้อความบนเมล์ที่ถูกส่งจะถูกเขียนบนล็อกแทย

Migrations & Seeding

คำอธิบายเบื้องต้น

Migrations คือการเก็บประวัติสร้างจุดเซฟของฐานข้อมูล. ทำให้เราสามารถเพิ่มลบตาราง โดยย้อนกลับได้หากไม่ถูกใจ ส่วนการเขียนตารางต้องไปดูเรื่อง Schema Builder การทำ migration จะเป็นการควบคุมการทำงานของ schema.

สร้าง Migrations

เริ่มด้วยการรันคำสั่ง `migrate:make` บน commandline:

การสร้างตารางโดยใช้ commandline

```
php artisan migrate:make create_users_table
```

ไฟล์ migration ถูกเก็บไว้ที่โฟลเดอร์ `app/database/migrations` แต่ละไฟล์จะมีวันกำกับชื่อด้วยเพื่อให้ระบบรู้ลำดับการสร้างไฟล์

เราสามารถกำหนดที่อยู่ของไฟล์ได้โดยพารามิเตอร์ `--path` เหมือนตัวอย่างข้างล่างครับ

```
php artisan migrate:make foo --path=app/migrations
```

พารามิเตอร์ `--table` และ `--create` ใช้ในการสร้าง ตารางทั้งคู่เลยครับ

```
php artisan migrate:make create_users_table --table=users --create
```

Running Migrations

การสั่งให้คำสั่ง migration ทำงาน

```
php artisan migrate
```

กำหนดที่อยู่ของไฟล์ที่จะรัน

```
php artisan migrate --path=app/foo/migrations
```

สั่งรันเฉพาะตรง package

```
php artisan migrate --package=vendor/package
```

> **หมายเหตุ:** ถ้าเราเจอ `error Class not found`ให้รันคำสั่ง `composer update`.

Rolling Back Migrations

การย้อนกลับการทำงานครั้งล่าสุด

```
php artisan migrate:rollback
```

การย้อนกลับทั้งหมด

```
php artisan migrate:reset
```

การย้อนกลับแล้วทำงานใหม่อีกรอบ

```
php artisan migrate:refresh

php artisan migrate:refresh --seed
```

Database Seeding

Laravel เตรียมการฟังก์ชันที่ช่วยในการป้อนข้อมูลจำลอง ที่อาจจะใช้ทดสอบการทำงานของเว็บไว้ในโฟลเดอร์ `app/database/seeds`. ชื่อคลาสเราก็ตั้งชื่อตามตารางและรูปแบบให้เป็นไปตามแบบนี้ครับ `UserTableSeeder`, โดยค่าเริ่มต้นแล้วจะเป็น `DatabaseSeeder` โดยใช้ฟังก์ชัน `call` เพื่อรันคลาสอื่นๆ ทำให้เราลำดับการทำงานได้

ตัวอย่าง

```
class DatabaseSeeder extends Seeder {

    public function run()
    {
        $this->call('UserTableSeeder');

        $this->command->info('User table seeded!');
    }

}

class UserTableSeeder extends Seeder {

    public function run()
    {
        DB::table('users')->delete();

        User::create(array('email' => 'foo@bar.com'));
    }

}
```

การทำงานก็ใช้คำสั่ง `db:seed` command บน Artisan CLI:

```
php artisan db:seed
```

ทำการย้อนกลับการทำงานด้วย

```
php artisan migrate:refresh --seed
```

Package Development

คำอธิบายเบื้องต้น

Packages คือการนำไลบรารี่ภายนอกเข้ามาใช้งานใน laravel ซึ่งส่วนมากจะอยู่บน Packagist
และส่วนมากอีกไม่ได้ปรับแต่งให้เข้ากับ framework ไหนเป็นพิเศษ ซึ่งตรงนี้เราก็ต้อง ออกแรงกันหน่อยครับ
ถ้าอยากจะมี package เป็นของเราเอง ส่วน การติดตั้ง package ถ้ามาเขียนในนี้คงจะยาว ผมเลยเขียนไว้ที่เว็บแล้ว
อยากรู้ต้องทำไงต้องตามไปดูเลยครับ

การสร้าง Package

สถานที่ๆเราจะทำการพัฒนาคือโฟลเดอร์ชื่อ `workbench` ครับ แต่ตอนแรกเราต้องเข้าไปตั้งค่าที่ไฟล์
`app/config/workbench.php` ในไฟล์นี้เราต้องเปลี่ยน `name` และ `email` ซึ่ง `composer.json` จะนำไปใส่เวลาเราสร้าง package
ขึ้นมา

สร้างโครงของ package ขึ้นมาด้วย CommandLine

```
php artisan workbench vendor/package --resources
```

parameter แรก `vendor` คือชื่อผู้พัฒนา `package` คือชื่อของ `package` `resources` เป็นพารามิเตอร์ที่บอกให้สร้าง `migrations`,
`views`, `config`, ด้วย

เมื่อคำสั่งข้างบนทำงาน `package`ของเราจะไปปรากฏที่โฟลเดอร์ `workbench` ต่อมาเราจะสร้าง `ServiceProvider`
ซึงการตั้งชื่อจะเป็นแบบนี้ `[Package]ServiceProvider` ส่วนการนำไปลงทะเบียนที่ไฟล์ `app.php`
นั้นเส้นทางของไฟล์จะเป็นแบบนี้ครับ `Taylor\Zapper\ZapperServiceProvider` ซึ่งต้องนำไปวางไว้ที่อาเรย์ชื่อ `providers`

ก่อนที่จะเริ่มพัฒนา package ของเรา ควรมารู้จักโครงสร้างของมันก่อนครับ

โครงสร้างของ package

โครงสร้างโฟลเดอร์ของ package

```
/src
    /Vendor
        /Package
            PackageServiceProvider.php
    /config
    /lang
    /migrations
    /views
/tests
/public
```

เมื่อตามเข้าไปดูในโฟลเดอร์ `src/[illegible]/[illegible]package[illegible]` จะเจอกับไฟล์ `ServiceProvider`กับโฟลเดอร์ `config`, `lang`, `migrations`, และ `views` ซึ่งจะเหมือนในตัวโฟลเดอร์ app หลักของเราแต่ย่อส่วนลงมาครับ

Service Providers

เริ่มแรกในไฟล์ `Service Provider` จะมีฟังก์ชัน `boot` และ `register`. ซึ่งในนี้เราจะสามารถ
ใส่ฟังก์ชันอะไรก็ได้ที่จะช่วยจัดการ package ของเรา เช่น ดึงไฟล์ route เข้ามา,ใช้ IOC ดึงคลาสอื่นเข้ามาช่วยเพิ่ม event เข้ามาดักฟัง

ฟังก์ชัน `register` จะเริ่มทำงานทันทีเมื่อ package เริ่มทำงาน, ถ้าเรา package ของเราต้องการคลาสอื่นๆ ในการทำงานร่วมด้วยต้องใช้เมทอด `boot` ครับ

โดยค่าเริ่มต้นแล้วฟังก์ชัน `boot` จะมีค่ามาให้ดังนี้

```
$this->package('vendor/package');
// [illegible] boot package
ClassLoader::addDirectories(array(
        app_path().'/widgets'
    ));
```

ถ้าลบบรรทัดแรกออกไปผลก็คือ package ไม่ทำงานครับ laravel จะไม่รู้จัก package นี้เลย

Package Conventions

ตัวอย่างการเรียกใช้งานค่าต่างๆใน package

เรียก view ของ package

```
return View::make('package::view.name');
```

เรียกค่าของการตั้งค่า

```
return Config::get('package::group.option');
```

> **หมายเหตุ:** ถ้า package ของเรามี migrations ควรใส่ชื่อ package เข้าไปอยู่ไฟล์ด้วย เพื่อป้องกันการเหมือนกับ packages อื่น

ขั้นตอนในการพัฒนา

พอสร้างโปรเจคเสร็จ เราก็ใช้คำสั่ง `php artisan dump-autoload` เพื่อสร้างไฟล์ที่บอกที่อยู่ของ package ของเราขึ้นมา

ตัวอย่าง

```
php artisan dump-autoload
```

Package Routing

การที่จะโหลดไฟล์ route เข้ามาใช้ต้องใช้ฟังก์ชัน `include` ในฟังก์ชัน `boot` ครับ

ตัวอย่างการดึง route มาใช้ใน Service Provider

```
public function boot()
{
    $this->package('vendor/package');
```

```
    include __DIR__.'/../../routes.php';
}
```

> **หมายเหตุ:**ถ้าต้องการใช้ controller ด้วยมั่นใจว่ามันต้องถูกเพิ่มไว้ในไฟล์ `composer.json` ในส่วน autoload

การตั้งค่า Package

บาง package เราต้องดึงค่าต่างๆ เช่น appid,appsecret มาใช้ โดยเริ่มต้นไฟล์ `config.php` จะอยู่ในตัว package กรณีที่ดึงลงมาด้วย composer แล้วติดตั้งเสร็จไฟล์ จะอยู่ที่โฟลเดอร์ `/app/config/package`

การเข้าถึงไฟล์ config

```
Config::get('package::file.option');
```

ตัวพารามิเตอร์แรกคือชื่อของ package หลังจาก semicolon คือชื่อไฟล์ หลังเครื่องหมายดอทคือชื่อ อาเรย์ `config.php`. ถ้าโฟลเดอร์ที่ใช้เก็บไฟล์ config ของ package มีไฟล์เดียวก็ไม่ต้องใส่ชื่อก็ได้ครับ

ตัวอย่างกรณีโฟลเดอร์มีไฟล์เดียว

```
Config::get('package::option');
```

บางครั้งไฟล์ที่ต้องใช้ใน package มันก็ดันไมได้อยู่ใน package สามารถใช้ฟังก์ชัน `addNamespace` ซึ่งมีอยู่ในคลาส `View`, `Lang`, กับ `Config`ดังตัวอย่างเลยครับ

```
View::addNamespace('package', __DIR__.'/path/to/views');
```

ทีนี้เราก็สามารถใช้งานคลาส View ใน package ได้ละ

```
return View::make('package::view.name');
```

Cascading Configuration Files

เมื่อเราดาวน์โหลดไฟล์ package มาด้วย composer ต้องนำไฟล์โฟลเดอร์ config ของ package ออกมาใช้ก่อน

ตัวอย่างการใช้งาน

```
php artisan config:publish vendor/package
```

เมื่อคำสั่งทำงานเสร็จจะปรากฎอยู่ที่ `app/config/packages/vendor/package`

Package Migrations

เราสามารถทำ migration ให้กับ package ได้โดยใช้พารามิเตอร์ `--bench` ตามตัวอย่างเลยครับ

สร้าง migration ให้ package ในโฟลเดอร์ workbench

```
php artisan migrate:make create_users_table --bench="vendor/package"
```

สั่งให้ migration ของ package ทำงาน

```
php artisan migrate --bench="vendor/package"
```

เมื่อพัฒนาเสร็จอยากจะลองใช้งานจริงเองก่อนเราต้องใช้พารามิเตอร์ `--package`เพื่อส่ง package ไปไว้ที่โฟลเดอร์ `vendor`

ตัวอย่างคำสั่งการย้าย package ไปไว้ที่ vendor

```
php artisan migrate --package="vendor/package"
```

Package Assets

บาง package จะมีไฟล์ JavaScript, CSS, และ images.เราไม่สามารถสร้างลิ้งโดยตรงจากโฟลเดอร์ `vendor` หรือ `workbench` ฉะนั้นเราต้องย้ายไปไว้ที่ โฟลเดอร์ `public`

การย้ายไฟล์ css,js,image โดยใช้ commandline

```
php artisan asset:publish
```

```
php artisan asset:publish vendor/package
```

แต่ถ้า package ยังอยู่ในโฟลเดอร์ workbench ต้องใช้คำสั่งนี้ครับ

```
php artisan asset:publish --bench="vendor/package"
```

ตอนนี้โฟลเดอร์จะถูกส่งไปไว้ที่ โฟลเดอร์`public/packages` ตัวอย่างชื่อโฟลเดอร์ `userscape/kudos` ตัวอย่างเส้นทางที่อยู่ `public/packages/userscape/kudos`.

การแบ่งปัน Packages ของเรา

เมื่อพัฒนาเสร็จแล้ว เราเกิดอยากให้คนอื่นช่วยพัฒนาต่อ หรือช่วยหาบั๊กให้ เราต้องอัพ package ของเราขึ้น github หรือ bitbucket ก่อนครับ หลังจากนั้นไปสมัคร Packagist เพื่อนำ package ของเราไปผูกไว้กับฐานข้อมูลหลักของ composer หลังจากนั่น ก็ใช้ composer โหลดลงมาใช้เลย

Pagination

คือแบ่งการแสดงข้อมูลเป็นหน้าไปครับ

การตั้งค่า

การตั้งค่าการของอยู่ที่ `app/config/view.php` ตัวแปรชื่อ `pagination` ในการแบ่งหน้า ฟังก์ชัน `pagination::slider` ใช้ในการสร้างเลขหน้า `pagination::simple` ใช้สร้างปุ่ม "previous" และ "next"

การใช้งาน

การใช้งานมีอยู่หลายรูปแบบ แต่ที่ง่ายที่สุดใช้เมทอด `paginate`บน query builder หรือ Eloquent model.

ตัวอย่างโดยใช้ query builder

```
$users = DB::table('users')->paginate(15);
```

You may also paginate Eloquent models:

ตัวอย่างโดยใช้ Eloquent Model

```
$users = User::where('votes', '>', 100)->paginate(15);
```

ในตัวอย่างเรากำหนดจำนวนข้มูลต่อหน้าได้ ส่วนการแสดงผลบน view เราจะใช้ฟังก์ชัน `links`

```
<div class="container">
    <?php foreach ($users as $user): ?>
        <?php echo $user->name; ?>
    <?php endforeach; ?>
</div>

<?php echo $users->links(); ?>
```

เพียงแค่นี้ก็จะได้การแบ่งหน้าละครับ.

เราสามารถจัดการๆ แบ่งหน้าได้โดยฟังก์ชันต่อไปนี้ครับ:

- `getCurrentPage`
- `getLastPage`
- `getPerPage`
- `getTotal`
- `getFrom`
- `getTo`

บางครั้งเราอยากสร้างเองเพราะอาจจะมีข้อมูลที่ต้องผ่านการคำนวนหลายขั้น ก็ใช้เมทอดนี้เลยครับ `Paginator::make`

ตัวอย่าง

```
$paginator = Paginator::make($items, $totalItems, $perPage);
```

Appending To Pagination Links

เราสามารถทำการเรียงลำดับการแสดงผลได้โดยใช้เมทอด `appends` เหมือนในตัวอย่าง

```
<?php echo $users->appends(array('sort' => 'votes'))->links(); ?>
```

ลิ้งที่ออกมาหน้าตาจะเป็นแบบนี้

```
http://example.com/something?page=2&sort=votes
```

Query Builder

คือการจัดการๆคิวรี่ของ laravel ช่วยอำนวยความสะดวกให้เรา ไม่ต้องเขียนคิวรี่ยาวๆ ด้วยตัวเองครับ

การเลือกข้อมูล

ดึงค่าทั้งหมดจากตาราง users

```
$users = DB::table('users')->get();

foreach ($users as $user)
{
    var_dump($user->name);
}
```

ดึงค่าแถวแรกจากตาราง users โดย name เท่ากับ john

```
$user = DB::table('users')->where('name', 'John')->first();

var_dump($user->name);
```

ดึงค่าจากตาราง users โดย name เท่ากับ john แล้วก็เอาแค่คอลัมน์ที่ชื่อว่า name

```
$name = DB::table('users')->where('name', 'John')->pluck('name');
```

ดึงค่าทั้งหมดจากตาราง roles โดยเอาแค่คอลัมน์ title

```
$roles = DB::table('roles')->lists('title');
```

ค่าที่คืนมาจะเป็นอาเรย์นะครับ ถ้าเราอยากใส่คีย์ให้แต่ละแถวเราใส่พารามิเตอร์ตัวที่สองเข้าไป name จะไปเป็นคีย์ให้กับ title

```
$roles = DB::table('roles')->lists('title', 'name');
```

เมทอด select ใช้กำหนดคำสั่งในการเลือกเอง

```
$users = DB::table('users')->select('name', 'email')->get();

$users = DB::table('users')->distinct()->get();

$users = DB::table('users')->select('name as user_name')->get();
```

เลือกข้อมูลจากผลการคิวรี่อีกที

```
$query = DB::table('users')->select('name');

$users = $query->addSelect('age')->get();
```

การใช้ where

```
$users = DB::table('users')->where('votes', '>', 100)->get();
```

การใช้หลายๆเงื่อนไขโดยวิธีเชนเมทอด

```
$users = DB::table('users')
                    ->where('votes', '>', 100)
                    ->orWhere('name', 'John')
                    ->get();
```

ใช้ between

```
$users = DB::table('users')
                    ->whereBetween('votes', array(1, 100))->get();
```

ตัวอย่างการใช้ where กับ In ร่วมกัน

```
$users = DB::table('users')
                    ->whereIn('id', array(1, 2, 3))->get();

$users = DB::table('users')
                    ->whereNotIn('id', array(1, 2, 3))->get();
```

ใช้หาแถวที่เป็นค่าว่างตามคอลัมน์

```
$users = DB::table('users')
                    ->whereNull('updated_at')->get();
```

ตัวอย่างการเรียงลำดับข้อมูล

```
$users = DB::table('users')
                    ->orderBy('name', 'desc')
                    ->groupBy('count')
                    ->having('count', '>', 100)
                    ->get();
```

การจำกัดข้อมูล

```
$users = DB::table('users')->skip(10)->take(5)->get();
```

Joins

ตัวอย่างการ join ครับ

```
DB::table('users')
            ->join('contacts', 'users.id', '=', 'contacts.user_id')
            ->join('orders', 'users.id', '=', 'orders.user_id')
            ->select('users.id', 'contacts.phone', 'orders.price');
```

การจอยแบบเติมคิวรี่ลงไปช่วยประหยัดเวลา เวลาคิดไม่ออกว่าจะใช้ฟังก์ไหนดี แทรกคิวรี่ลงไปตรงๆ เลย:

```
DB::table('users')
        ->join('contacts', function($join)
        {
            $join->on('users.id', '=', 'contacts.user_id')->orOn(...);
        })
        ->get();
```

Advanced Wheres

การ where แบบ หลายเงื่อนไข

```
DB::table('users')
            ->where('name', '=', 'John')
            ->orWhere(function($query)
            {
                $query->where('votes', '>', 100)
                      ->where('title', '<>', 'Admin');
            })
            ->get();
```

หน้าตาของคิวรี่จะออกมาเป็นแบบนี้

```
select * from users where name = 'John' or (votes > 100 and title <> 'Admin')
```

หาว่ามีค่านี้อยู่ไหม

```
DB::table('users')
            ->whereExists(function($query)
            {
                $query->select(DB::raw(1))
                      ->from('orders')
                      ->whereRaw('orders.user_id = users.id');
            })
            ->get();
```

หน้าตาของคิวรี่จะออกมาเป็นแบบนี้

```
select * from users
where exists (
    select 1 from orders where orders.user_id = users.id
)
```

Aggregates การหาผลรวมชนิดต่างๆ

ตัวอย่างการคำนวนค่า

```
$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');
```

Raw Expressions

Raw คือการใส่คิวรี่แบบสดๆ เข้าไปเลยไม่ต้องไปให้ฟังก์ชันสร้างให้ ประหยัดเวลามากขึ้นเมื่อเราต้องการค้นหาแบบซับซ้อน

ตัวอย่างการใช้คำสั่งคิวรี่

```
$users = DB::table('users')
                     ->select(DB::raw('count(*) as user_count, status'))
                     ->where('status', '<>', 1)
                     ->groupBy('status')
                     ->get();
```

การเพิ่มและลดค่าให้คอลัมน์

```
DB::table('users')->increment('votes');

DB::table('users')->increment('votes', 5);

DB::table('users')->decrement('votes');

DB::table('users')->decrement('votes', 5);
```

การกำหนดค่าคอลัมน์ที่จะเพิ่มค่าให้

```
DB::table('users')->increment('votes', 1, array('name' => 'John'));
```

การเพิ่มข้อมูล

ตัวอย่าง

```
DB::table('users')->insert(
    array('email' => 'john@example.com', 'votes' => 0)
);
```

การเพิ่มค่าพร้อมกับเพิ่มค่า id ด้วย

```
$id = DB::table('users')->insertGetId(
    array('email' => 'john@example.com', 'votes' => 0)
);
```

> **หมายเหตุ:** ถ้าใช้ PostgreSQL เมทอด insertGetId คาดหวังว่าจะใช้คอลัมน์ "id" เป็นตัวที่มันจะเพิ่มให้

การเพิ่มหลายๆข้อมูล

```
DB::table('users')->insert(array(
    array('email' => 'taylor@example.com', 'votes' => 0),
    array('email' => 'dayle@example.com', 'votes' => 0),
));
```

การแก้ไข

ตัวอย่าง

```
DB::table('users')
            ->where('id', 1)
            ->update(array('votes' => 1));
```

การลบ

ตัวอย่าง

```
DB::table('users')->where('votes', '<', 100)->delete();
```

การลบข้อมูลทั้งหมดเป็นการลบแบบแถวต่อแถว

```
DB::table('users')->delete();
```

การลบแบบลบทั้งตารางด้วยแล้วสร้างขึ้นใหม่

```
DB::table('users')->truncate();
```

การทำ Unions

union คือการจับผลลัพท์ของการ selecet 2 ครั้ง มารวมกันเป็นหนึ่งผลลัพท์ **Performing A Query Union**

```
$first = DB::table('users')->whereNull('first_name');

$users = DB::table('users')->whereNull('last_name')->union($first)->get();
```

เมทอด `unionAll` ใช้งานเหมือนกับ `union` เลยครับ

Caching Queries

เราสามารถทำการแคชหรือบันทึกผลการคิวรี่ไว้บน session ก่อนด้วยเมทอด `remember`

ตัวอย่าง

```
$users = DB::table('users')->remember(10)->get();
```

ในตัวอย่างเราจะแคชผลการค้นหานี้เป็นเวลา 10 นาที ระหว่างนี้การคิวรี่จากตัวอย่างจะไม่ไปดึงข้อมูลจากฐานข้อมูล แต่จะดึงจากแคชจนกว่าจะหมดเวลาครับ

Queues

คลาสที่ช่วยเรียงลำดับการทำงานของฟังก์ชันต่างๆ

การตั้งค่า

Laravel Queue เตรียมฟังก์ชันที่ใช้ในการเข้าถึง api ของเว็บที่ให้บริการคิวไว้ การคิวคือการเรียงลำดับงานของเว็บไซต์ เช่น เรามีเมล์ที่ต้องส่งถึง 1000 ฉบับถ้าส่งแบบเดิม server อาจจะรับไม่ไหว เราจึงมีคลาส queue มาเพื่อการนี้ครับ

ไฟล์ที่ใช้ตั้งค่าเก็บไว้ที่ `app/config/queue.php`. ในไฟล์จะมีข้อมูลที่เราต้องใช้ในการเชื่อมต่อผู้ให้บริการคิว เช่น Beanstalkd, IronMQ, Amazon SQS, and synchronous (สำหรับการทดสอบในเครื่อง)

ชื่อคลาสของผู้ให้บริการคิวที่เราจะป้อนเข้าไป:

- Beanstalkd: `pda/pheanstalk`
- Amazon SQS: `aws/aws-sdk-php`
- IronMQ: `iron-io/iron_mq`

การใช้งานเบื้องต้น

ในการส่งงานใหม่เข้าไปในคิวเราใช้ฟังก์ชัน `Queue::push`

Pushing A Job Onto The Queue

```
Queue::push('SendEmail', array('message' => $message));
```

พารามิเตอร์ตัวแรกเป็น ฟังก์ชันที่เราใช้ควบคุมคิวนี้. ตัวที่สองเป็นข้อมูลที่เราจะทำการคิว

ตัวอย่างฟังก์ชันที่ใช้ควบคุมการคิว

```
class SendEmail {
```

```
    public function fire($job, $data)
    {
        //
    }

}
```

ฟังก์ชัน`fire`รับพารามิเตอร์ `Job` ตัวที่สองคือข้อมูล `data` ที่จะส่งลงคิว

ถ้าเราไม่ใช่เมทอด `fire`เราสามารถกำหนดได้ตามตัวอย่างครับ

เปลี่ยนจาก fire เป็น Push

```
Queue::push('SendEmail@send', array('message' => $message));
```

เมื่อมีการทำงานไปแล้วเราก็ต้องลบข้อมูลออกไปโดยใช้เมทอด `delete` เพื่อลบ `Job` instance:

ตัวอย่างการลบ

```
public function fire($job, $data)
{
    // Process the job...

    $job->delete();
}
```

ถ้าเราต้องการเอางานที่ทำไปกลับมาเข้าคิวอีกครั้งก็ใช้เมทอด `release` ครับ

ตัวอย่าง

```
public function fire($job, $data)
{
    // Process the job...

    $job->release();
}
```

เราสามารถกำหนดเวลาที่จะหน่วงไว้ก่อนที่จะสั่งให้งานต่อไปทำงานได้แบบนี้ครับ

```
$job->release(5);
```

เมื่องานที่เข้าคิวเกิดข้อผิดพลาดขึ้น จะถูกนำกลับไปต่อคิวใหม่ เราสามารถตรวจสอบการทำงานใหม่ได้โดยฟังก์ชัน `attempts`

ตรวจหางานที่มีการพยายามทำมากกว่า 3 ครั้ง

```
if ($job->attempts() > 3)
{
    //
}
```

การเรียกข้อมูลของงาน

```
$job->getJobId();
```

Queueing Closures

เราสามารถใช้งานฟังก์ชันที่ไม่มีชื่อในการสร้างงานได้โดยตัวอย่างเลยครับ

```
Queue::push(function($job) use ($id)
{
    Account::delete($id);

    $job->delete();
});
```

> **หมายเหตุ:** เมื่อใช้ฟังก์ชันที่ไม่มีชื่อกับ queue ตัวแปร `__DIR__` และ `__FILE__` จะไม่สามารถใช้งานได้

ถ้าใช้บริการของ Iron.io push queues, ควรจะไม่ใช้งานฟังก์ชันที่ไม่มีชื่อในเมล์. จะมีการตรวจสอบคำร้องว่าส่งมาจาก Iron.io.จริงหรือไม่ ตัวอย่าง `https://yourapp.com/queue/receive?token=SecretToken`.ควรทำการตรวจสอบค่า secrettoken ก่อนทำการคิว.

Running The Queue Listener

Laravel เตรียมคำสั่ง`queue:listen` เพื่อรอรับคำขอที่มากจากผู้ให้บริการ

ตัวอย่างการใช้งาน

```
php artisan queue:listen
```

เราสามารถกำหนดค่าการเชื่อมต่อเสริมลงไปได้:

```
php artisan queue:listen connection
```

เราสามารถใช้โปรแกรม Supervisor เพื่อตรวจสอบว่าคำสั่ง queue listener ยังทำงานอยู่ไหม

สามารถตั้งค่าเวลาที่จะอนุญาตุให้เกิดทำงานขึ้นได้

ตัวอย่างการหน่วงเวลา

```
php artisan queue:listen --timeout=60
```

สั่งให้งานชิ้นแรกที่อยู่บนคิวทำงาน

```
php artisan queue:work
```

Push Queues

Push queues คือการโยนภาระการรอรับคิวจากที่ทำบนเครื่องขิงเรา เปลี่ยนไปให้ผู้บริการทำแทน ซึ่งมี Iron.io เจ้าเดียวที่สนับสนุน. ก่อนอื่นเราต้องไปสมัครบริการของ Ironio ก่อน, แล้วนำข้อมูลการเชื่อมต่อมาใส่ไว้ที่ `app/config/queue.php`

ต่อมาก็ใช้คำสั่ง Artisan `queue:subscribe` เพื่อลงทะเบียนคิวไว้กับ Iron.io:

ตัวอย่าง

```
php artisan queue:subscribe queue_name http://foo.com/queue/receive
```

ตอนนี้ ถ้าเราไปดูที่หน้าแสดงผลคิวที่ Iron.io , ก็จะเห็นรายการคิว และรายชื่อลิ้งที่เราจะสั่งให้ทำงาน. เราสามารถลงทะเบียนไว้ทีละหลายๆลิ้งก็ได้.ต่อมาเราต้องมาสร้าง route เพื่อรับคำร้องขอของ Iron.io โดยชื่อ route จะเป็น `queue/receive` และส่งค่ากลับไปด้วยฟังก์ชัน `Queue::marshal`

```
Route::post('queue/receive', function()
{
    return Queue::marshal();
});
```

เมทอด `marshal` จะดูแลการทำงานของคิวให้

Redis

Redis เป็นโอเพ่นซอร์สฐานข้อมูลขนาดเล็ก ที่เก็บข้อมูลในรูปแบบ คีย์กับค่า หนึ่งค่าต้องมีคีย์กำกับ ประมาณนี้ครับ อยากรู้จักมากกว่านี้ต้องลองไปเล่นที่ try redis โดยข้อมูลที่อยู่ในนี้จะมี strings, hashes, lists, sets, และ sorted sets.

การปรับแต่ง

การปรับแต่ง redis เราทำใน **app/config/database.php** ในนี้เราจะเห็นอาเรย์ชื่อ **redis** แบบนี้ครับ

```
'redis' => array(

    'cluster' => true,

    'default' => array('host' => '127.0.0.1', 'port' => 6379),

),
```

โดยค่าเริ่มต้นแล้ว ถูกตั้งให้สนับสนุนการพัฒนาบนเครื่อง เราสามารถปรับแต่งได้ตามใจครับ อาเรย์ชื่อ `cluster` ใช้เพื่อบอกให้ redis ที่อยู่ในเครื่องทำการ client-side sharding ข้าม Redis nodes ของคุณ, ทำให้เราสามารถสร้าง ram จำนวนมากได้. แต่การทำ client-side sharding แต่เราไม่สามารถจับการ failover ได้

การใช้งาน

เราสามารถสร้างตัวแปรที่เป็นตัวแทนของเมทอด `Redis::connection` โดยการ

```
$redis = Redis::connection();
```

เรายังไม่ได้กำหนดชื่อ server redis ของเราต้องกำหนดก่อนนะครับ

```
$redis = Redis::connection('other');
```

ถ้าอยากศึกษาต่อไปที่นี้ครับ Redis commands ในการจัดการต่างๆ laravel มีฟังก์ชันต่างๆ มาให้แล้วครับ:

```
$redis->set('name', 'Taylor');

$name = $redis->get('name');

$values = $redis->lrange('names', 5, 10);
```

ถ้าไม่ต้องการใช้คำสั่ง laravel ก็สามารถใช้เมทอด `command` ในการใช้คำสั่งของ redis ตรงๆได้

```
$values = $redis->command('lrange', array(5, 10));
```

คลาส `Redis` ที่เป็นตัว static คลาสครับ:

```
Redis::set('name', 'Taylor');

$name = Redis::get('name');

$values = Redis::lrange('names', 5, 10);
```

Pipelining

Pipelining คือการส่งคำสั่งหลายๆ ตัวไปยัง redis server laravel มีเมทอด `pipeline` มาให้ใช้

ตัวอย่าง

```
Redis::pipeline(function($pipe)
{
    for ($i = 0; $i < 1000; $i++)
    {
        $pipe->set("key:$i", $i);
    }
});
```

Requests & Input

คลาสนี้ใช้จัดการคำร้องขอต่างๆ เช่น ค่าที่ส่งมาจากฟอร์ม

Basic Input

รับเฉพาะค่าที่มีชื่อตรงกับที่กำหนด

```
$name = Input::get('name');
```

กำหนดค่าสำรองกรณีค่าที่ส่งมาเป็น null

```
$name = Input::get('name', 'Sally');
```

ตรวจว่าค่าที่ส่งมาเป็นค่าว่างไหม

```
if (Input::has('name'))
{
    //
}
```

รับค่าทั้งหมดของคำร้องขอ

```
$input = Input::all();
```

รับค่าเป็นกรณีๆ ไป

```
$input = Input::only('username', 'password');

$input = Input::except('credit_card');
```

Cookies

การดึงค่าจาก cookies

```
$value = Cookie::get('name');
```

สร้าง cookie และส่งคืนไปให้ผู้ใช้งาน

```
$response = Response::make('Hello World');

$response->withCookie(Cookie::make('name', 'value', $minutes));
```

สร้าง cookie ที่ไม่หมดอายุ

```
$cookie = Cookie::forever('name', 'value');
```

Old Input

คือการที่เราเก็บค่าที่ได้จากฟอร์มไว เพื่อทำการใส่ในฟอร์มถัดไป หรือนำไปแสดงหลังจากหน้าโหลดเสร็จ

การนำค่าในฟอร์มเก็บใส่ session

```
Input::flash();
```

การรับค่าเฉพาะกรณี

```
Input::flashOnly('username', 'email');

Input::flashExcept('password');
```

เราทำการส่งค่าในฟอร์มเก่าที่เก็บไว้ขึ้นไปในฟอร์ม

```
return Redirect::to('form')->withInput();

return Redirect::to('form')->withInput(Input::except('password'));
```

การดึงค่าจากในฟอร์มเก่า

```
Input::old('username');
```

Files

รับค่าจากการอัพโหลดไฟล์

```
$file = Input::file('photo');
```

ตรวจว่าไฟล์ถูกอัพโหลดไหม

```
if (Input::hasFile('photo'))
{
    //
}
```

ค่าที่ถูกคืนมาจะเป็นเมทอด `file` method ซึ่งมาจากคลาส `Symfony\Component\HttpFoundation\File\UploadedFile` ซึ่งสืบทอดมาจากคลาส `SplFileInfo` ซึ่งเตรียมเมทอดไว้ให้เราจัดการไฟล์ไว้เยอะเลยครับ

เปลี่ยนที่อยู่ให้ไฟล์ที่อัพขึ้นมา

```
Input::file('photo')->move($destinationPath);

Input::file('photo')->move($destinationPath, $fileName);
```

ดึงค่าเส้นทางทีอยูของไฟล์

```
$path = Input::file('photo')->getRealPath();
```

ดึงชื่อเริ่มต้นของไฟล์

```
$name = Input::file('photo')->getClientOriginalName();
```

ดึงค่าขนาดของไฟล์

```
$size = Input::file('photo')->getSize();
```

ดึงนามสกุลของไฟล์

```
$mime = Input::file('photo')->getMimeType();
```

Request Information

คลาส `Request`ของ laravel สืบทอดมาจากคลาส `Symfony\Component\HttpFoundation\Request` ต่อไปนี้คือฟังก์ชันสำคัญครับ

ดึงค่า URI จากการเรียกครั้งล่าสุด

```
$uri = Request::path();
```

ตรวจสอบว่าคำร้องขอที่ส่งเข้ามาตรงกับกฎที่เราตั้งไว้ไหม

```
if (Request::is('admin/*'))
{
    //
}
```

ดึงค่า url ของคำร้องขอ

```
$url = Request::url();
```

ดึงค่า URI เฉพาะส่วน

```
$segment = Request::segment(1);
// http::/taqmaninw.com/admin/post/id?=3
// $sengment ������� admin
```

**ดึงค่า header **

```
$value = Request::header('Content-Type');
```

ดึงค่าจากตัวแปร $_SERVER

```
$value = Request::server('PATH_INFO');
```

ตรวจว่าคำขอเป็น ajax ไหม

```
if (Request::ajax())
{
    //
}
```

ตรวจว่าคำขอมาจาก https ไหม

```
if (Request::secure())
{
    //
}
```

Views & Responses

บทนี้จะมาพูดถึงคลาส views กับ Response นะครับ

Basic Responses

การส่งค่าคืนแบบง่ายๆ

```
Route::get('/', function()
{
    return 'Hello World';
});
```

สร้างการส่งกลับเอง

คลาส `Response` สืบทอดมาจากคลาส `Symfony\Component\HttpFoundation\Response`
เราจะมาดูเฉพาะเมทอดที่สำคัญกันนะครับ

ตัวอย่างการสร้างคำตอบกลับนะครับ

```
$response = Response::make($contents, $statusCode);

$response->header('Content-Type', $value);

return $response;
```

เพิ่ม cookie ลงไปในคำตอบกลับ

```
$cookie = Cookie::make('name', 'value');

return Response::make($content)->withCookie($cookie);
```

Redirects การส่งกลับ

ส่งกลับไปที่ route

```
return Redirect::to('user/login');
```

ส่งกลับไปพร้อมกับ ข้อความ

```
return Redirect::to('user/login')->with('message', 'Login Failed');
```

ส่งกลับไปที่ route ที่มีชื่อย่อตามตัวอย่าง

```
return Redirect::route('login');
```

ส่งกลับไปที่ route ที่มีชื่อย่อตามตัวอย่างพร้อมกับค่า

```
return Redirect::route('profile', array(1));
```

ส่งกลับไปที่ route ที่มีชื่อย่อตามตัวอย่างพร้อมกับตัวแปร

```
return Redirect::route('profile', array('user' => 1));
```

**ส่งกลับไปที่ฟังก์ชันใน controller **

```
return Redirect::action('HomeController@index');
```

ส่งกลับไปที่ฟังก์ชันใน controller พร้อมกับพารามิเตอร์

```
return Redirect::action('UserController@profile', array(1));
```

ส่งกลับไปที่ฟังก์ชันใน controller พร้อมกับตัวแปร

```
return Redirect::action('UserController@profile', array('user' => 1));
```

Views

Views คือส่วนที่ใช้เก็บไฟล์ที่ใช้สร้างหน้า html นะครับจะถูกเก็บไว้ที่โฟลเดอร์ `app/views` ตัวอย่าง view

```
<!-- View stored in app/views/greeting.php -->

<html>
    <body>
        <h1>Hello, <?php echo $name; ?></h1>
    </body>
</html>
```

การใช้งาน view เบื้องต้นครับ

```
Route::get('/', function()
{
    return View::make('greeting', array('name' => 'Taylor'));
});
```

ส่งค่าไปที่ view ครับ

```
$view = View::make('greeting', $data);

$view = View::make('greeting')->with('name', 'Steve');
```

ในตัวอย่างตัวแปร `$name`จะถูกใช้งานบน View ได้

You may also share a piece of data across all views:

```
View::share('name', 'Steve');
```

การส่ง view แทรกเข้าไปในอีก view หนึ่งครับ

เราสร้างโฟลเดอร์ขึ้นมาเก็บ view ที่เราจะทำเป็น view ย่อยก่อนตัวอย่าง `app/views/child/view.php` ตัวอย่างการใช้งาน

```
$view = View::make('greeting')->nest('child', 'child.view');

$view = View::make('greeting')->nest('child', 'child.view', $data);
```

ผลที่ออกมาครับ

```
<html>
    <body>
        <h1>Hello!</h1>
        <?php echo $child; ?>
    </body>
</html>
```

View Composers

View composers คือเมทอดที่ช่วยเราในการจัดการค่าที่เราต้องแสดง ในทุกหน้าของ view ลดการเขียนโค้ดซ้ำซ้อน

ตัวอย่างการใช้งาน

```
View::composer('profile', function($view)
{
    $view->with('count', User::count());
});
```

ตอนนี้ทุกครั้งที่ `profile` view ถูกสร้าง `count` จะถูกส่งขึ้นไปด้วย

เราสามารถส่งขึ้นไปทีละหลายๆ view ได้

```
View::composer(array('profile','dashboard'), function($view)
{
    $view->with('count', User::count());
});
```

ถ้าเราต้องการทำให้เป็นคลาสเพื่อง่ายต่อการจัดกลุ่ม เราต้องทำแบบนี้ครับ

```
View::composer('profile', 'ProfileComposer');
```

สร้างคลาสขึ้นมา

```
class ProfileComposer {
```

```
    public function compose($view)
    {
        $view->with('count', User::count());
    }

}
```

แล้วอย่าลืมเพิ่มเข้าไปที่ไฟล์ `composer.json`

การส่งกลับแบบพิเศษ

สร้างการส่งกลับในรูปแบบของ json

```
return Response::json(array('name' => 'Steve', 'state' => 'CA'));
```

สร้างการส่งกลับในรูปแบบของ jsonp

```
return Response::json(array('name' => 'Steve', 'state' => 'CA'))->setCallback(Input::get('callback'));
```

สร้างการส่งกลับในรูปแบบของการดาวน์โหลดไฟล์

```
return Response::download($pathToFile);

return Response::download($pathToFile, $name, $headers);
```

Routing

ใช้ในการกำหนดว่าเมื่อเราเรียกลิ้งนี้จะให้ทำอะไรขึ้นบ้าง

Basic Routing

ในการตั้งค่าเราจะไปที่ `app/routes.php` โดยรูปแบบของฟังก็ชันที่เป็น Route จะเป็นแบบ Closure callback Closure คืออะไรตามไปตามเข้าไปอ่าน ที่นี่ครับ

การรับค่าที่เป็น get

```
Route::get('/', function()
{
    return 'Hello World';
});
```

การรับค่าที่เป็น POST

```
Route::post('foo/bar', function()
{
    return 'Hello World';
});
```

กำหนด route ในการเรียกพารามิเตอร์ foo ในทุกรูปแบบเมทอด

```
Route::any('foo', function()
{
    return 'Hello World';
});
```

ลิ้งที่เรียกมาต้องเป็น https เท่านั้น

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

```
    return 'Must be over HTTPS';
}));
```

Route Parameters

ตัวอย่างการกำหนดรูปแบบของพารามิเตอร์

```
Route::get('user/{id}', function($id)
{
    return 'User '.$id;
});
```

พารามิเตอร์แบบมีหรือไม่มีก็ได้

```
Route::get('user/{name?}', function($name = null)
{
    return $name;
});
```

กำหนดพารามิเตอร์แบบตายตัว

```
Route::get('user/{name?}', function($name = 'John')
{
    return $name;
});
```

การใช้ regex ตรวจสอบว่าพารามิเตอร์ตรงกับที่กำหนดไว้ไหม

```
Route::get('user/{name}', function($name)
{
    //
})
->where('name', '[A-Za-z]+');

Route::get('user/{id}', function($id)
{
    //
})
```

```
->where('id', '[0-9]+');
```

จะใส่ไปเป็นอาเรย์ก็ได้

```
Route::get('user/{id}/{name}', function($id, $name)
{
    //
})
->where(array('id' => '[0-9]+', 'name' => '[a-z]+'))
```

Route Filters

คือการกำหนดฟังก์ชันที่ใช้ในการตรวจสอบข้อมูล `auth` ใช้ตรวจว่ามีการล็อกอินไหม, `guest` ตรวจว่ายังไม่ได้ล็อกอิน, และ `csrf`ตรวจว่าเป็นการทำ csrf ไหม.ซึ่งเราจะไปประกาศไว้ที่ `app/filters.php`

ตัวอย่างการสร้าง filter

```
Route::filter('old', function()
{
    if (Input::get('age') < 200)
    {
        return Redirect::to('home');
    }
});
```

การใส่ filter ให้ route

```
Route::get('user', array('before' => 'old', function()
{
    return 'You are over 200 years old!';
}));
```

การใส่ route หลายตัว

```
Route::get('user', array('before' => 'auth|old', function()
{
    return 'You are authenticated and over 200 years old!';
```

```
}));
```

การกำหนดค่าเฉพาะให้ filter

```
Route::filter('age', function($route, $request, $value)
{
    //
});

Route::get('user', array('before' => 'age:200', function()
{
    return 'Hello World';
}));
```

filter บางตัวเราสั่งให้ทำงานหลังจากที่ route ทำงานไปแล้วเราต้องกำหนดตัวแปร `$response` เพื่อกำหนดค่าที่จะส่งไปให้ตัวฟังก์ชันด้วย

```
Route::filter('log', function($route, $request, $response, $value)
{
    //
});
```

Pattern Based Filters

เราสามารถกำหนด filter ให้ทำงานเฉพาะเมื่อมีการเรียกตรงกับที่เรากำหนดได้ ตามตัวอย่างเลยครับ.

```
Route::filter('admin', function()
{
    //
});

Route::when('admin/*', 'admin');
```

ตามตัวอย่างเราเพิ่ม filter ชื่อ `admin` เข้ากับทุกลิ้งที่มี `admin/` อยู่ข้างใน

แล้วก็ยังสามารถกำหนดเมทอดให้ได้ด้วย

```
Route::when('admin/*', 'admin', array('post'));
```

Filter Classes

ในการกรองขั้นสูงเราสามารถสร้างคลาสขึ้นมาได้เอง แล้วทำการใช้ IoC Container เรียกใช้คลาสนั้น

ตัวอย่างคลาส

```
class FooFilter {

    public function filter()
    {
        // Filter logic...
    }

}
```

ลงทะเบียนคลาสโดยให้ชื่อที่จะนำไปใช้ว่า foo

```
Route::filter('foo', 'FooFilter');
```

Named Routes

คือการตั้งชื่อย่อให้กับ route:

```
Route::get('user/profile', array('as' => 'profile', function()
{
    //
}));
```

กำหนดให้ชื่อย่อนี้จะใช้ controller ไหน

```
Route::get('user/profile', array('as' => 'profile', 'uses' => 'UserController@showProfile'));
```

ตอนนี้เราใช้ชื่อย่อ เพื่อสร้างลิ้งได้แล้ว

```
$url = URL::route('profile');

$redirect = Redirect::route('profile');
```

เราใช้เมทอด`currentRouteName` เพื่อดึงชื่อของ Route ที่ทำงานในขณะนี้ได้

```
$name = Route::currentRouteName();
```

Route Groups

เราสามารถกำหนดกลุ่มให้ Route ได้ทำให้สะดวกมากขึ้น

```
Route::group(array('before' => 'auth'), function()
{
    Route::get('/', function()
    {
        // Has Auth Filter
    });

    Route::get('user/profile', function()
    {
        // Has Auth Filter
    });
});
```

Sub-Domain Routing

การสร้าง Route ให้กับโดนเมนย่อย

```
Route::group(array('domain' => '{account}.myapp.com'), function()
{

    Route::get('user/{id}', function($account, $id)
    {
        //
    });

});
```

Route Prefixing

ในการกำหนดคำที่ใช้กำหนดกลุ่มของ เราใช้ prefix ในการตรวจสอบ

ตัวอย่างการใช้ prefix

```
Route::group(array('prefix' => 'admin'), function()
{

    Route::get('user', function()
    {
        //
    });

});
```

Route Model Binding

คือการผูกโมเดลเขาไปกับ Route โดยใช้เมทอด `Route::model`

การใช้งาน

```
Route::model('user', 'User');
```

ต่อมาก็กำหนดให้เมื่อมีการเรียกลิ้งที่มี `{user}`เป็นพารามิเตอร์

```
Route::get('profile/{user}', function(User $user)
{
    //
});
```

เราก็จะทำการแทรก `User` instance เข้าไปใน Route ยกตัวอย่าง `profile/1` ถูกเรียก `User` instance ก็จะมี ID = 1.

ถ้าพารามิเตอร์ที่ส่งเข้ามาไม่ตรงกับ `model` ใดๆเราสามารถกำหนดการแสดงข้อผิดพลาดำได้

```
Route::model('user', 'User', function()
{
    throw new NotFoundException;
});
```

ต่อมา เมทอด `Route::bind` เป็นการผูกพารามิเตอร์เข้ากับ โมเดล เมื่อมีการส่งค่าเข้าตรงกับ route ที่กำหนดค่าก็จะก็จะถูกส่งมาที่เมทอดนี้ก่อน

```
Route::bind('user', function($value, $route)
{
    return User::where('name', $value)->first();
});
```

Schema Builder

คลาส `Schema`ใช้ในการจัดการตาราง มักใช้ร่วมกับคำสั่ง migration

สร้างและลบตาราง

สร้างตารางใหม่ด้วยเมทอด `Schema::create` ตามตัวอย่างเลยครับ

```
Schema::create('users', function($table)
{
    $table->increments('id');
});
```

เมทอดตัวแรกคือชื่อตาราง พารามิเตอร์ที่สองคือ `Closure` ที่จะรับออปเจคของคลาส `Blueprint` ในการสร้างตาราง

จะเปลี่ยนชื่อตารางใช้เมทอด`rename`

```
Schema::rename($from, $to);
```

ในการเลือกการเชื่อมต่อกรณีเรามีหลายฐานข้อมูลใช้เมทอด `Schema::connection`

```
Schema::connection('foo')->create('users', function($table)
{
    $table->increments('id'):
});
```

จะลบตารางใช้เมทอด `Schema::drop`

```
Schema::drop('users');

Schema::dropIfExists('users');
```

เพิ่มคอลัมน์

ในการแก้ไขตารางเราใช้คำสั่ง `Schema::table`

```
Schema::table('users', function($table)
{
    $table->string('email');
});
```

ตารางคำสั่งต่างของคลาส `Blueprint` ครับ

Command	Description
`$table->increments('id');`	ใช้ทำ auto_increment ให้ (primary key).
`$table->string('email');`	ค่าที่ออกมาจะเป็น VARCHAR ความยาว 255
`$table->string('name', 100);`	ค่าที่ออกมาจะเป็น VARCHAR ความยาว 100
`$table->integer('votes');`	ค่าที่ออกมาจะเป็น interger
`$table->bigInteger('votes');`	ค่าที่ออกมาจะเป็น bigint
`$table->smallInteger('votes');`	ค่าที่ออกมาจะเป็นSMALLINT
`$table->float('amount');`	ค่าที่ออกมาจะเป็น FLOAT
`$table->decimal('amount', 5, 2);`	ค่าที่ออกมาจะเป็น decimal ที่มีค่าระหว่าง 5,2
`$table->boolean('confirmed');`	ค่าที่ออกมาจะเป็น BOOLEAN
`$table->date('created_at');`	ค่าที่ออกมาจะเป็น DATE
`$table->dateTime('created_at');`	ค่าที่ออกมาจะเป็นDATETIME
`$table->time('sunrise');`	ค่าที่ออกมาจะเป็น TIME

`$table->timestamp('added_on');`	ค่าที่ออกมาจะเป็น TIMESTAMP
`$table->timestamps();`	เพิ่มคอลัมน์ **created_at** และ **updated_at** columns
`$table->softDeletes();`	เพิ่มคอลัมน์ **deleted_at**
`$table->text('description');`	ค่าที่ออกมาจะเป็น TEXT
`$table->binary('data');`	ค่าที่ออกมาจะเป็น BLOB
`$table->enum('choices', array('foo', 'bar'));`	ค่าที่ออกมาจะเป็น ENUM
`->nullable()`	อนุญาตให้เป็นค่าว่างได้
`->default($value)`	สร้างค่าเริ่มต้น
`->unsigned()`	เลี่ยน INTEGER เป็น UNSIGNED

ถ้าเราใช้ MySQL dสามารถใช้เมทอด `after` ทำการเรียงลำดับคอลัมน์

ตัวอย่าง

```
$table->string('name')->after('email');
```

เปลี่ยนชื่อ Columns

ใช้เมทอด `renameColumn` ครับ

ตัวอย่าง

```
Schema::table('users', function($table)
{
    $table->renameColumn('from', 'to');
```

```
});
```

> **หมายเหตุ:** คอลัมน์ชนิด `enum` ไม่สนับสนุน

ลบ Columns

ตัวอย่าง

```
Schema::table('users', function($table)
{
    $table->dropColumn('votes');
});
```

ลบหลายๆ คอลัมน์

```
Schema::table('users', function($table)
{
    $table->dropColumn('votes', 'avatar', 'location');
});
```

ตรวจว่าตารางมีอยู่ไหม

ตรวจว่าตารางมีอยู่ไหม

```
if (Schema::hasTable('users'))
{
    //
}
```

ตรวจว่าคอลัมน์มีอยู่ไหม

```
if (Schema::hasColumn('users', 'email'))
{
    //
}
```

เพิ่ม Indexes

สร้างทั้ง Column และ Index

```
$table->string('email')->unique();
```

Command	Description
`$table->primary('id');`	เพิ่ม primary key
`$table->primary(array('first', 'last'));`	เพิ่ม composite keys
`$table->unique('email');`	เพิ่ม unique index
`$table->index('state');`	เพิ่ม basic index

Foreign Keys คีย์เชื่อม

ตัวอย่างการเพิ่มคีย์เชื่อม

```
$table->foreign('user_id')->references('id')->on('users');
```

ในตัวอย่างเราทำการให้คอลัมน์ `user_id`อ้างอิงกับคอลัมน์ `id` บนตาราง`users`

เราสามารถเสริมคำสั่ง "on delete" และ "on update" เข้าไปเหมือนตัวอย่าง

```
$table->foreign('user_id')
      ->references('id')->on('users')
      ->onDelete('cascade');
```

ในการลบใช้เมทอด `dropForeign` ตัวอย่าง

```
$table->dropForeign('posts_user_id_foreign');
```

> **หมายเหตุ:** ให้ชนิดของคอลัมน์ที่เป็นคีย์เชื่อมให้เป็น `unsigned` ทุกครั้งกรณีที่เชื่อมไปยัง คอลัมน์ที่เป็น interger และเป็น auto_increment ครับ

ลบ Indexes

ในการลบคีย์เชื่อม larave ตั้งค่าคีย์เป็นค่าเริ่มต้นให้แล้วนะครับ โดยอ้างอิงจากชนิดกับชื่อของตาราง

Command	Description
`$table->dropPrimary('users_id_primary');`	ลบคีย์หลักจากตาราง users
`$table->dropUnique('users_email_unique');`	ลบคีย์เดี่ยวจากตาราง users
`$table->dropIndex('geo_state_index');`	ลบคีย์ทั่วไปจาก ตาราง geo

ชนิดของตาราง

การเซตชนิดของตารางเราใช้เมทอด `engine` ตามตัวอย่างครับ

```
Schema::create('users', function($table)
{
    $table->engine = 'InnoDB';

    $table->string('email');
});
```

Security

คลาสนี้ใช้ในการสร้างระบบรักษาความปลอดภัยต่างอย่างเช่น การเข้ารหัสเพื่อใช้ใน password,session,cookie

การเก็บรหัสผ่าน

Class `Hash` ของ `laravel` ใช้ส่วนขยาย `Bcrypt` ของ php มาพัฒนาต่อยอด

การสร้างค่า hash

```
$password = Hash::make('secret');
```

การตรวจสอบค่า hash

```
if (Hash::check('secret', $hashedPassword))
{
    // The passwords match...
}
```

ตรวจว่า password ต้องการเข้ารหัสอีกครั้ง กรณีลืมรหัสผ่าน

```
if (Hash::needsRehash($hashed))
{
    $hashed = Hash::make('secret');
}
```

การยืนยันตัวบุคคล

การล็อกอิน laravel เตรียมเมทอด `Auth::attempt` มาให้ตัวอย่างการใช้งาน

```
if (Auth::attempt(array('email' => $email, 'password' => $password)))
{
    return Redirect::intended('dashboard');
}
```

ค่า `email` เราสามารถเปลี่ยนไปตามใจเราได้ครับ ส่วนเมทอด `Redirect::intended` ใช้ส่งผู้ใช้งานกลับไปที่ลิ้งที่เข้าเรียกมาครับ

เมื่อเมทอด `attempt` ถูกเรียก event `auth.attempt` จะถูกเรียกและ event `auth.login` จะถูกเรียกเมื่อการเข้าสู่ระบบสำเร็จ

ตรวจสอบว่ามีการล็อกอินค้างอยู่ไหม

```
if (Auth::check())
{
    // The user is logged in...
}
```

ตัวอย่างการปรับปรุงเมทอด attempt ให้สามารถทำการจำชื่อผู้ใช้กับรหัสผ่านได้

```
if (Auth::attempt(array('email' => $email, 'password' => $password), true))
{
    // The user is being remembered...
}
```

กากำหนดเงื่อนไขตอนล็อกอิน

```
if (Auth::attempt(array('email' => $email, 'password' => $password, 'active' => 1)))
{
    // �� active ��� 1 �����������������
}
```

การเข้าถึงเข้อมูลคนที่ล็อกอิน

```
$email = Auth::user()->email;
```

ใช้เมทอด `loginUsingId` เพื่อดึง Id ของคนที่ล็อกอินมา

```
Auth::loginUsingId(1);
```

สมมุตว่าเรากำหนดว่าการเปลี่ยนแปลงเลขบัตรเครดิตต้องใช้รหัสผ่านยืนยันเพิ่มเติม เมทอด `validate`

สามารถทำงานนี้ให้เราได้

การใช้งานการยืนยันตัวตนโดยไม่ได้เข้าสู่ระบบ

```
if (Auth::validate($credentials))
{
    //
}
```

การล็อกอินแบบไม่มี session หรือ cookies

```
if (Auth::once($credentials))
{
    //
}
```

เมทอดที่ใช้ล็อกเอาท์

```
Auth::logout();
```

การจำลองการล็อกอิน

การล็อกอินแบบ ที่เราจำลองขึ้นมาเองครับ เมทอด`login` ใช้ค่าจากฐานข้อมูลมาล็อกอินได้ทันทีเลย

```
$user = User::find(1);

Auth::login($user);
```

การป้องกัน CSRF

ทำการแนบค่า token เข้ากับฟอร์ม

```
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
```

ตรวจสอบค่า token ที่ถูกส่งมา

```
Route::post('register', array('before' => 'csrf', function()
{
    return 'You gave a valid CSRF token!';
}));
```

HTTP Basic Authentication

laravel เตรียม filter ชื่อ `auth.basic` เพื่อตรวจว่า มีการล็อกอินไหม

ตัวอย่าง

```
Route::get('profile', array('before' => 'auth.basic', function()
{
    // Only authenticated users may enter...
}));
```

โดยค่าเริ่มต้นแล้วเมทอด `basic` ใช้คอลัมน์ email ในการตรวจสอบ ถ้าเราจะเปลี่ยนก็ใช้

```
return Auth::basic('username');
```

laravel เตรียมฟังก์ชัน `Oncebasic` มาเพื่อการล็อกอินแบบไม่สร้าง session ไว้เหมาะกับการให้ผู้ใช้งานใช้ในกรณีไปล็อกอินเครื่องที่ไม่ใช่ของตัวเอง

ตัวอย่าง

```
Route::filter('basic.once', function()
{
    return Auth::onceBasic();
});
```

การจัดการการลืมรหัสผ่าน

การลืมรหัสผ่านและการสร้างใหม่

ส่งรหัสผ่านใหม่

laravel เตรียมการมาให้เราสามารถสร้างระบบการส่และเปลี่ยนรหัสผ่าน ให้เราโดยการให้ `User` model ทำการสืบทอด `Illuminate\Auth\Reminders\RemindableInterface`.

การใช้งาน RemindableInterface

```
class User extends Eloquent implements RemindableInterface {

    public function getReminderEmail()
    {
        return $this->email;
    }

}
```

ต่อมาเราก็ต้องสร้างตารางให้ระบบลืมรหัสผ่านก่อนโดย `php artisan auth:reminders`

สร้างตัว migration ของตารางลืมรหัส

```
php artisan auth:reminders

php artisan migrate
```

การส่งรหัสใช้เมทอด `Password::remind`

ตัวอย่างการใช้งาน

```
Route::post('password/remind', function()
{
    $credentials = array('email' => Input::get('email'));

    return Password::remind($credentials);
});
```

> **หมายเหตุ:** เราต้องสร้าง view ที่ชื่อ `auth.reminder.email`เพื่อรับ email เองนะครับ

เราสามารถส่งข้อความเพิ่มเติมให้ผู้ใช้โดยส่งพารามิเตอร์ไป `$message` ไปในฟังก์ชัน `remind`

```
return Password::remind($credentials, function($message, $user)
{
    $message->subject('Your Password Reminder');
});
```

โดยค่าเริ่มต้นแล้วเมทอด `remind` จะส่งกลับมาที่หน้าที่เรียกใช้ ถ้าเกิดข้อผิดพลาดขึ้น ตัวแปร `error`จะมีค่าขึ้นใน session ส่วนเมื่อสำเร็จตัวแปร `success` ก็จะปรากฏขึ้นมาใน session แทน หน้าตาของหน้า `auth.reminder.email` ควรเป็นแบบนี้ครับ

```
@if (Session::has('error'))
    {{ trans(Session::get('reason')) }}
@elseif (Session::has('success'))
    An e-mail with the password reset has been sent.
@endif

<input type="text" name="email">
<input type="submit" value="Send Reminder">
```

การรีเซตรหัสผ่าน

การสร้าง route เพื่อรับการที่ผู้ใช้งานกดลิ้งทำการรีเซตรหัสผ่าน

```
Route::get('password/reset/{token}', function($token)
{
    return View::make('auth.reset')->with('token', $token);
});
```

หน้า view ที่ทำการให้ผู้ใช้งานทำการเปลี่ยนรหัสผ่าน

```
@if (Session::has('error'))
    {{ trans(Session::get('reason')) }}
@endif

<input type="hidden" name="token" value="{{ $token }}">
<input type="text" name="email">
<input type="password" name="password">
<input type="password" name="password_confirmation">
```

ตัวอย่างการสร้าง route เพื่อทำการรับค่ารหัสผ่านใหม่

```
Route::post('password/reset/{token}', function()
{
    $credentials = array('email' => Input::get('email'));

    return Password::reset($credentials, function($user, $password)
    {
        $user->password = Hash::make($password);

        $user->save();

        return Redirect::to('home');
    });
});
```

ถ้าการเปลี่ยนรหัสผ่านสำเร็จ `User` instance และรหัสผ่านใหม่จะถูกเก็บลงฐานข้อมูลและ ส่งกลับไปหน้า `home`

Encryption

Laravel เตรียมการเข้ารหัสแบบ AES-256 โดยส่วนเสริม mcrypt ของ PHP มาให้แล้ว

กาเข้ารหัส

```
$encrypted = Crypt::encrypt('secret');
```

> **Note:** มั่นใจว่าเราเปลี่ยนค่า `key` ตรงที่ `app/config/app.php` ไม่งั้นการเข้ารหัสจะไม่ค่อยปลอดภัยครับ

การถอดรหัส

```
$decrypted = Crypt::decrypt($encryptedValue);
```

การกำหนดรูปแบบต่างๆ

```
Crypt::setMode('ctr');
```

```
Crypt::setCipher($cipher);
```

Session

การตั้งค่าเบื้องต้น

ไฟล์ที่ใช้ตั้งค่าจะอยู่ที่ `app/config/session.php`.โดยชนิดของ session จะมีหลายชนิดนะครับแต่โดยเริ่มต้นแล้วจะเป็น `native` ส่วนการตั้งค่าอื่นๆ ก็จะเป็นเวลาที่จะให้ seesion มีชีวิตอยู่ ที่อยู่ของ seesion ชื่อของ cookie และอื่นๆ ครับ

การใช้งาน

การสร้างค่าแล้วเก็บใน session

```
Session::put('key', 'value');
```

ดึงค่าจาก Session

```
$value = Session::get('key');
```

ดึงค่าเริ่มต้นของ session

```
$value = Session::get('key', 'default');

$value = Session::get('key', function() { return 'default'; });
```

ตรวจว่ามีค่านี้ใน Session หรือไม่

```
if (Session::has('users'))
{
    //
}
```

ลบค่าออกจาก Session

```
Session::forget('key');
```

ลบค่าทั้งหมด Session

```
Session::flush();
```

สร้าง Session ID อีกครั้ง

```
Session::regenerate();
```

Flash Data

หลายๆครั้งเราต้องฝากค่าไว้ใน session เพื่อนำไปใช้ในการทำงานต่อไป สามารถใช้เมทอด `Session::flash` ตัวอย่าง

```
Session::flash('key', 'value');
```

ทำการเรียกใช้ falsh message อีกครั้ง

```
Session::reflash();
```

ทำการเรียกใช้งานอีกครั้งเฉพาะค่า

```
Session::keep(array('username', 'email'));
```

การเก็บ session ในฐานข้อมูล

เมื่อเราใช้ฐานข้อมูลเก็บ session เราต้องสร้างตารางขึ้นมาก่อน ด้วยคำสั่ง`Schema` ดังตัวอย่าง

```
Schema::create('sessions', function($table)
{
    $table->string('id')->unique();
    $table->text('payload');
    $table->integer('last_activity');
});
```

ตอนนี้เราก็ใช้คำสั่ง `php artisab session:table` เป็นอันจบครับ

Templates

Controller Layouts

เราใช้ตัวแปร `layout` เพื่อกำหนดเลเอาท์หลักให้กับ controller ครับ

ตัวอย่างการใช้งาน

```
class UserController extends BaseController {

    /**
     * The layout that should be used for responses.
     */
    protected $layout = 'layouts.master';

    /**
     * [illegible]
     */
    public function showProfile()
    {
        $this->layout->content = View::make('user.profile');
    }

}
```

Blade Templating

Blade template คือคลาสที่ใช้ในการเขียน html ขึ้นมาด้วย php ทำให้เราสามารถสร้าง html ที่มีโครงสร้างซับซ้อนได้ การจะใช้นั้นต้องตั้งชื่อไฟล์เป็น `view.blade.php`

ตัวอย่างที่ยังใช้ html อยู่

```
<!-- Stored in app/views/layouts/master.blade.php -->

<html>
```

```
    <body>
        @section('sidebar')
            This is the master sidebar.
        @show

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

ใช้ blade ในการสร้างเลเอาท์ทั้งหมด

```
@extends('layouts.master')

@section('sidebar')
    @parent

    <p>This is appended to the master sidebar.</p>
@stop

@section('content')
    <p>This is my body content.</p>
@stop
```

`extend` ใช้ในการดึงค่าจากเลเอาท์อื่นมาใช้ `@parent` ทำให้เราสามารถใช้ view อื่นแทรกเข้ามาได้ `@section` ก็ใช้งานโดยการแทรก html จากไฟล์อื่นเข้าไป

การใช้งานฟังก์ชัน php ใน blade

การแสดงข้อมูล การแทรกข้อมูล

```
Hello, {{ $name }}.

The current UNIX timestamp is {{ time() }}.
```

ทีนี้เราก็ใช้ syntax แบบสั้นๆ ในการแสดงผลจาก server ได้แล้ว

```
Hello, {{{ $name }}}.
```

การใช้ if

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif

@unless (Auth::check())
    You are not signed in.
@endunless
```

การใช้ Loops

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

การแทรก Views

```
@include('view.name')
```

การแสดงภาษา

```
@lang('language.line')
```

```
@choice('language.line', 1);
```

การทำคอมเม้น

```
{{-- This comment will not be in the rendered HTML --}}
```

Unit Testing

Laravel สร้างขึ้นมาด้วยแนวคิดของการทดสอบเป็นเบื้องต้นอยู่แล้วครับ โดยหลักแล้วจะสนับสนุนไลบราลี่ PHPUnit เป็นพื้นฐาน และ `phpunit.xml`ไฟล์ได้ถูกเตรียมการเอาไว้ให้แล้ว. Laravel เตรียมคลาส Symfony HttpKernel, DomCrawler, และ BrowserKit components ที่อนุญาตให้เราจำลองบราวเซอร์ขึ้นมาและเข้าไปแก้ไขไฟล์ html ได้ ตัวอย่างไฟล์อยู่ที่โฟลเดอร์ `app/tests`

Defining & Running Tests

การสร้างไฟล์สำหรับทดสอบนั้นเราจะไปสร้างที่โฟลเดอร์ `app/tests` สร้างคลาสที่สืบทอดคลาส `TestCase`.

ตัวอย่างคลาสสำหรับใช้ทดสอบ

```
class FooTest extends TestCase {

    public function testSomethingIsTrue()
    {
        $this->assertTrue(true);
    }

}
```

เราจะทำการทดสอบโดยรัน `phpunit` บน commandline

> **หมายเหตุ:** ถ้าคุณประกาศเมทอด `setUp` มั่นใจว่าได้เรียก `parent::setUp`แล้ว

สภาวะการตั้งค่าสำหรับการทดสอบ

เมื่อใช้งาน unit tests, Laravel จะทำการเปลี่ยนการสภาวะการตั้งค่าให้ไปเป็น `testing`. และจะตัดการทำงานของ `session` และ `cache` หมายความว่าจะไม่มีแคชและ session เกิดขึ้นระหว่างการทดสอบ

การเรียก Routes ในขณะทดสอบ

ตัวอย่างการเรียก Route ในขณะทำการทดสอบ

```
$response = $this->call('GET', 'user/profile');

$response = $this->call($method, $uri, $parameters, $files, $server, $content);
```

เราสามารถตรวจสอบออปเจค `Illuminate\Http\Response`

```
$this->assertEquals('Hello World', $response->getContent());
```

ตัวอย่างการเรียก Controller ในขณะทดสอบ

```
$response = $this->action('GET', 'HomeController@index');

$response = $this->action('GET', 'UserController@profile', array('user' => 1));
```

เมทอด `getContent` จะส่งค่าเป็นตัวอักษรกลับคืนมา `View` เราสามารถเข้าถึงได้ด้วยตัวแปร `original`

```
$view = $response->original;

$this->assertEquals('John', $view['name']);
```

ถ้าจะเรียก HTTPS route,เราต้องใช้เมทอด`callSecure`

```
$response = $this->callSecure('GET', 'foo/bar');
```

DOM Crawler

คลาส DOM Crawler ทำให้เราสามารถตรวจสอบ html ที่ถูกสร้างขึ้นมาระหว่างการทดสอบได้ ตัวอย่างการใช้

```
$crawler = $this->client->request('GET', '/');

$this->assertTrue($this->client->getResponse()->isOk());

$this->assertCount(1, $crawler->filter('h1:contains("Hello World!")'));
```

Mocking Facades

เมื่อเราทำการทดสอบ,เราจะทำการจำลองในการเรีกคลาส Facade ตัวอย่างเราจะทำการเรียก controller

```
public function getIndex()
{
    Event::fire('foo', array('name' => 'Dayle'));

    return 'All done!';
}
```

เราสามารถจำลองคลาส`Event`โดยใช้เมทอด `shouldReceive`

การจำลองคลาส Facade

```
public function testGetIndex()
{
    Event::shouldReceive('fire')->once()->with(array('name' => 'Dayle'));

    $this->call('GET', '/');
}
```

> **หมายเหตุ:** คุณไม่ควรจำลองคลาส `Facade Request` ใช้เมทอด `call` ดีกว่าครับ

Framework Assertions

เมทอด `assert` ใช้ในการตรวจสอบว่าค่าที่ออกมาตรงกับที่เราคาดหวังไว้ไหม

คาดหวังว่าค่าที่ส่งมาจะไม่ผิดพลาด

```
public function testMethod()
{
    $this->call('GET', '/');

    $this->assertResponseOk();
}
```

คาดหวังว่าจะเป็น 403

```
$this->assertResponseStatus(403);
```

คาดหวังว่าฟังก์ชันจะส่งกลับไปที่ route

```
$this->assertRedirectedTo('foo');

$this->assertRedirectedToRoute('route.name');

$this->assertRedirectedToAction('Controller@method');
```

คาดหวังว่าในหน้า view จะมีค่า

```
public function testMethod()
{
    $this->call('GET', '/');

    $this->assertViewHas('name');
    $this->assertViewHas('age', $value);
}
```

คาดหวังว่าใน session จะมีค่า

```
public function testMethod()
{
    $this->call('GET', '/');

    $this->assertSessionHas('name');
    $this->assertSessionHas('age', $value);
}
```

Helper Methods

คลาส `TestCase` มีเมทอดช่วยให้เราทำการทดสอบได้ง่ายๆ เยอะเลยครับ.

เมทอด `be`ใช้ในการจำลองการล็อกอิน

ตัวอย่าง

```
$user = new User(array('name' => 'John'));

$this->be($user);
```

ทำการเพิ่มข้อมูลลงฐานข้อมูลในขณะทดสอบ

```
$this->seed();

$this->seed($connection);
```

Validation

คือการตรวจสอบค่าต่างๆ ที่ป้อนเข้ามา หรือระหว่างการทำงานของฟังก์ชันต่างๆ โดยจะแสดงข้อผิดพลาดให้เราด้วย โดยคลาสที่ทำหน้าที่นั้นชื่อ Validator ครับ

การใช้งานเบื้องต้น

การใช้งานคลาส validator

```
$validator = Validator::make(
    array('name' => 'Dayle'),
    array('name' => 'required|min:5')
);
```

อาเรย์ตัวแรกคือข้อมูลที่เราจะทำการตรวจนั้นเอง ตัวที่สองคือรูปแบบที่เราต้องการ การใช้เครื่องหมาย | การตรวจสอบออกเป็นหลายๆ แบบ

ใช้อาเรย์ในการกำหนดกฎ

```
$validator = Validator::make(
    array('name' => 'Dayle'),
    array('name' => array('required', 'min:5'))
);
```

คลาส `Validator` จะสร้างเมทอด ชื่อ `fails` (หรือ `passes`) เพื่อตรวจสอบผล

```
if ($validator->fails())
{
    // The given data did not pass validation
}
```

ถ้าไม่ผ่านเราสามารถดึงข้อความแสดงข้อผิดพลาดได้.

```
$messages = $validator->messages();
```

เมทอด `failed` ใช้ในการเข้าถึงกฎที่เราตั้งไว้

```
$failed = $validator->failed();
```

การจัดการข้อความแสดงข้อผิดพลาด

เมื่อเรียกเมทอด `messages` บนตัว `Validator` instance,เราจะได้รับ `MessageBag` instance ที่จะมีเมทอดให้เราจัดการข้อความ

แสดงข้อความแสดงข้อผิดพลาดเฉพาะตัวแรก

```
echo $messages->first('email');
```

รับข้อความแสดงข้อผิดพลาดทั้งหมด

```
foreach ($messages->get('email') as $message)
{
    //
}
```

รับข้อความแสดงข้อผิดพลาดจากทุคอลัมน์

```
foreach ($messages->all() as $message)
{
    //
}
```

ตรวจว่ามีข้อความแสดงข้อผิดพลาดจากคอลัมน์ eamil ไหม

```
if ($messages->has('email'))
{
    //
}
```

รับข้อความแสดงข้อผิดพลาดโดยใส่รูปแบบให้ด้วย

```
echo $messages->first('email', '<p>:message</p>');
```

> **หมายเหตุ:** โดยเริ่มต้น, รูปแบบข้อความจะถูกจัดในรูปแบบที่นำไปใช้งานร่วมกับ twiiter bootstrap ได้.

รับข้อความแสดงข้อผิดพลาดทั้งหมดพร้อมใส่รูปแบบ

```
foreach ($messages->all('<li>:message</li>') as $message)
{
    //
}
```

การแสดงข้อความแสดงข้อผิดพลาดบน view

ตัวอย่างนี้เราจะส่งข้อความแสดงข้อผิดพลาด ไปให้ view

```
Route::get('register', function()
{
    return View::make('user.register');
});

Route::post('register', function()
{
    $rules = array(...);

    $validator = Validator::make(Input::all(), $rules);

    if ($validator->fails())
    {
        return Redirect::to('register')->withErrors($validator);
    }
});
```

ถ้าการตรวจสอบไม่ผ่านเราจะใช้เมทอด `withErrors` ส่งข้อความแสดงข้อผิดพลาด ขึ้นไปบน View ด้วย

ไม่ควรส่งข้อความแสดงข้อผิดพลาด ไปบน Route ทีเป็น method GET เพราะ laravel จะตรวจสอบข้อผิดพลาดบน session ทุกคำร้องขอ

เมื่อทำการรีไดเรคเราสามารถเข้าถึงข้อความแสดงข้อผิดพลาด โดยใชตัวแปร`$errors` ดังตัวอย่างครับ

```
<?php echo $errors->first('email'); ?>
```

กฏในการตรวจสอบที่ laravel เตรียมไว้

accepted

ค่าทีจะผ่านคือ *yes*, *on*, or *1*.เหมาะสำหรับใช้ในการตรวจสอบว่ายอมรับ "Terms of Service" ไหม

active_url

ตรวจสอบว่าลิ้งตายยัง โดยใช้ `checkdnsrr` ซึ่งเป็น PHP function.

after:_date_

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของเวลาหลักจากใช้ `strtotime` อยู่หลังสุดแปลงไหม

alpha

ตรวจสอบว่าค่าที่ส่งมาเป็นรูปแบบของตัวอักษรต่างๆ ไหม

alpha_dash

ตรวจสอบว่าค่าที่ส่งมาเป็นรูปแบบของตัวเลขที่มีเครื่องหมาย _ รวมอยู่ด้วยไหม

alpha_num

ตรวจสอบว่าค่าที่ส่งมาเป็นรูปแบบของตัวเลขไหม

before:_date_

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของเวลาหลักจากใช้ `strtotime` อยู่หน้าสุดไหม

between:_min,_*max*

ตรวจสอบว่าค่าที่ส่งมาเป็นมีค่าอยู่ระหว่าง min กับ max ไหม

confirmed

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของฟอร์มที่มีรูปแบบ ชื่อ_confirmation ยกตัวอย่างการ ตรวจสอบ `password`,ว่าตรงกับ`password_confirmation` ไหม

date

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของเวลาหลักจากใช้ `strtotime` ไหม.

date_format:_format_

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของเวลาที่กำหนดไหม

different:_field_

ค่าตรง *field* ต้องมีค่าต่างจากค่าที่ป้อนเข้ามาถึงจะผ่าน

email

ตรวจสอบว่าค่าที่ส่งมาเป็นมีรูปแบบของ email

exists:_table*,_column*

ฟอร์มที่อยู่ในการตรวจสอบต้องมีชื่อตรงกับคอลัมน์ในฐานข้อมูล

การใช้งานเบื้องต้น

```
'state' => 'exists:states'
```

การใช้งานโดยใส่ค่าที่ต้องการตรวจไปหลายค่า

```
'state' => 'exists:states,abbreviation'
```


เราสามารถกำหนดเงื่อนไขให้กฎคล้ายๆการทำคิวรี้ ครับ

```
'email' => 'exists:staff,email,account_id,1'
```

image

ตรวจสอบว่าค่าที่ส่งมาเป็นรูปภาพมีนามสกุล(jpeg, png, bmp, or gif) ไหม

in:_foo,_*bar*,...

ตรวจสอบว่าค่าที่ส่งมามีค่าตรงกับค่าใน foo,bar ไหม

integer

ตรวจสอบว่าค่าที่ส่งมามีรูปแบบของเลขจำนวนเต็มไหม

ip

ตรวจสอบว่าค่าที่ส่งมามีรูปแบบของ IP address.

max:_value_

ตรวจสอบว่าค่าที่ส่งมามีค่าน้อยกว่าค่าที่กำหนดไว้

mimes:_foo,_*bar*,...

ตรวจสอบว่าค่าที่ส่งมามีรูปแบบของ mime type ตรงกับที่กำหนดไหม

ตัวอย่างการตรวจสอบนามสกุลของไฟล์

```
'photo' => 'mimes:jpeg,bmp,png'
```

min:_value_

ตรวจสอบว่าค่าที่ส่งมามีจำนวนน้อยกว่าไหมถ้ามีน้อยกว่าก็ไม่ผ่าน

not_in:_foo,_*bar*,...

ตรวจสอบว่าค่าที่ส่งมามีค่าตรงกับค่าที่ตั้งไว้ไหม ถ้ามีก็ไม่ผ่านครับ

numeric

ตรวจสอบว่าค่าที่ส่งมาเป็นตัวเลขไหม

regex:_pattern_

ตรวจสอบว่าค่าที่ส่งมามีรูปแบบกับ regular expression ที่กำหนดไว้ไหม

required

ตรวจสอบว่าค่าที่ส่งมาเป็นค่าว่างไหม ถ้าเป็นก็ไม่ผ่านครับ

required_if:_field*,_value*

ตรวจว่าค่าใน *field* ต้องไม่ว่างและ ตรงกับ *value*

required_with:_foo*,_bar*,...

ตรวจว่าฟิล *foo* ต้องมีค่าหาก bar มีค่าด้วย

required_without:_foo*,_bar*,...

ตรวจว่าฟิล *foo* ต้องมีค่าหาก bar ไม่มีค่า

same:_field_

ตรวจว่าค่าที่ส่งเข้ามาซ้ำกับค่าที่กำหนดไว้ไหม

size:_value_

ตรวจว่าค่าที่ส่งเข้ามาตรงกับที่กำหนดไว้ไหม กรณีเป็นคำจะตรวจสอบจำนวนคำ เป็นตัวเลขก็เทียบตามค่า เป๊ไฟล์เทียบตามขนาดของไฟล์เป็นกิโลไบต์

unique:_table*,_column*,_except*,_idColumn*

ตรวจว่าค่าที่ส่งมาซ้ำกับในตารางไหม.

ตัวอย่างการใช้ตรวจว่าอีเมล์นี้มีในตาราง user ไหม

```
'email' => 'unique:users'
```

ตัวอย่างการใช้ตรวจว่าอีเมล์นี้มีในตาราง user ตรงคอลัมน์ email-address ไหม

```
'email' => 'unique:users,email_address'
```

ตัวอย่างการใช้ตรวจว่าอีเมล์นี้มีในตาราง user ตรงคอลัมน์ email-address ไหม โดยไม่สนใจ id ที่มีค่าเท่ากับ 10

```
'email' => 'unique:users,email_address,10'
```

url

ตรวจว่าค่าเป็น url ไหม

Custom Error Messages

เราสามารถปรับแต่งข้อความที่แสดงข้อผิดพลาดได้

ตัวอย่าง

```
$messages = array(
    'required' => 'The :attribute field is required.',
);

$validator = Validator::make($input, $rules, $messages);
```

การใช้ข้อความที่เรากำหนดร่วมกับ Place-Holders

```
$messages = array(
    'same'    => 'The :attribute and :other must match.',
    'size'    => 'The :attribute must be exactly :size.',
```

```
    'between' => 'The :attribute must be between :min - :max.',
    'in'      => 'The :attribute must be one of the following types: :values',
);
```

กำหนดข้อความให้แต่ละคอลัมน์เลย

```
$messages = array(
    'email.required' => 'We need to know your e-mail address!',
);
```

บางกรณีเราต้องการกำหนดข้อความที่แสดงให้เป็นเฉพาะแต่ละภาษาไป ซึ่งเราต้องไปเพิ่มที่อาเรย์ชื่อ `custom` ใน `app/lang/xx/validation.php` ตาม ภาษาที่ไป

ตัวอย่าง

```
'custom' => array(
    'email' => array(
        'required' => 'We need to know your e-mail address!',
    ),
),
```

การสร้างตัวตรวจสอบ

เราสามารถสร้างฟังก์ชันในการตรวจสอบได้เองโดย laravel เตรียมเมทอด `Validator::extend` มาเพื่อการนั้นครับ

ตัวอย่าง

```
Validator::extend('foo', function($attribute, $value, $parameters)
{
    return $value == 'foo';
});
```

ตัวอย่างข้างบนเรารับตัวแปรมาสามตัวครับ `$attribute` คือชื่อข้อมูลที่จะตรวจ `$value` ค่าของข้อมูล `$parameters` ค่าอื่นๆ

You may also pass a class and method to the `extend` method instead of a Closure:

```
Validator::extend('foo', 'FooValidator@validate');
```

มีอีกวิธีในการสร้างคลาสของเราเองโดยการสืบทอด `Illuminate\Validation\Validator` ทีนี้ฟังก์ชันต้องมีคำว่า `validate`นำหน้าด้วยนะครับ

ตัวอย่าง

```
<?php

class CustomValidator extends Illuminate\Validation\Validator {

    public function validateFoo($attribute, $value, $parameters)
    {
        return $value == 'foo';
    }

}
```

ต่อมาเราต้องเอาคลาสของเรามาลงทะเบียน

```
Validator::resolver(function($translator, $data, $rules, $messages)
{
    return new CustomValidator($translator, $data, $rules, $messages);
});
```

เราสามารถสร้างเมทอดที่ใช้ในการแสดงข้อผิดพลาดโดยตามรูปแบบนี้ครับ `replaceXXX` ตามตัวอย่าง

```
protected function replaceFoo($message, $attribute, $rule, $parameters)
{
    return str_replace(':foo', $parameters[0], $message);
}
```