Eloquent: Mutators

Introduction

Accessors and mutators allow you to format Eloquent attribute values when you retrieve or set them on model instances. For example, you may want to use the Laravel encrypter to encrypt a value while it is stored in the database, and then automatically decrypt the attribute when you access it on an Eloquent model.

In addition to custom accessors and mutators, Eloquent can also automatically cast date fields to Carbon instances or even cast text fields to JSON.

Accessors & Mutators

Defining An Accessor

To define an accessor, create a `getFooAttribute` method on your model where `Foo` is the "studly" cased name of the column you wish to access. In this example, we'll define an accessor for the `first_name` attribute. The accessor will automatically be called by Eloquent when attempting to retrieve the value of the `first_name` attribute:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the user's first name.
     *
     * @param  string  $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}
```

As you can see, the original value of the column is passed to the accessor, allowing you to manipulate and return the value. To access the value of the accessor, you may access the `first_name` attribute on a model instance:

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

$firstName = $user->first_name;
```

You may also use accessors to return new, computed values from existing attributes:

```
/**
 * Get the user's full name.
 *
 * @return string
 */
public function getFullNameAttribute()
{
    return "{$this->first_name} {$this->last_name}";
}
```

If you would like these computed values to be added to the array / JSON representations of your model, you will need to append them.

Defining A Mutator

To define a mutator, define a `setFooAttribute` method on your model where `Foo` is the "studly" cased name of the column you wish to access. So, again, let's define a mutator for the `first_name` attribute. This mutator will be automatically called when we attempt to set the value of the `first_name` attribute on the model:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Set the user's first name.
     *
     * @param  string  $value
     * @return void
     */
    public function setFirstNameAttribute($value)
    {
        $this->attributes['first_name'] = strtolower($value);
    }
}
```

The mutator will receive the value that is being set on the attribute, allowing you to manipulate the value and set the manipulated value on the Eloquent model's internal `$attributes` property. So, for example, if we attempt to set the `first_name` attribute to `Sally`:

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

$user->first_name = 'Sally';
```

In this example, the `setFirstNameAttribute` function will be called with the value `Sally`. The mutator will then apply the `strtolower` function to the name and set its resulting value in the internal `$attributes` array.

Date Mutators

By default, Eloquent will convert the `created_at` and `updated_at` columns to instances of Carbon, which extends the PHP `DateTime` class and provides an assortment of helpful methods. You may add additional date attributes by setting the `$dates` property of your model:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that should be mutated to dates.
     *
     * @var array
     */
    protected $dates = [
        'seen_at',
    ];
}
```

You may disable the default `created_at` and `updated_at` timestamps by setting the public `$timestamps` property of your model to `false`.

When a column is considered a date, you may set its value to a UNIX timestamp, date string (`Y-m-d`), date-time string, or a `DateTime` / `Carbon` instance. The date's value will be correctly converted and stored in your database:

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

$user->deleted_at = now();

$user->save();
```

As noted above, when retrieving attributes that are listed in your `$dates` property, they will automatically be cast to Carbon instances, allowing you to use any of Carbon's methods on your attributes:

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

return $user->deleted_at->getTimestamp();
```

Date Formats

By default, timestamps are formatted as `'Y-m-d H:i:s'`. If you need to customize the timestamp format, set the `$dateFormat` property on your model. This property determines how date attributes are stored in the database, as well as their format when the model is serialized to an array or JSON:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The storage format of the model's date columns.
     *
     * @var string
     */
    protected $dateFormat = 'U';
}
```

Attribute Casting

The `$casts` property on your model provides a convenient method of converting attributes to common data types. The `$casts` property should be an array where the key is the name of the attribute being cast and the value is the type you wish to cast the column to. The supported cast types are: `integer`, `real`, `float`, `double`, `decimal:<digits>`, `string`, `boolean`, `object`, `array`, `collection`, `date`, `datetime`, and `timestamp`. When casting to `decimal`, you must define the number of digits (`decimal:2`).

To demonstrate attribute casting, let's cast the `is_admin` attribute, which is stored in our database as an integer (`0` or `1`) to a boolean value:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'is_admin' => 'boolean',
    ];
}
```

Now the `is_admin` attribute will always be cast to a boolean when you access it, even if the underlying value is stored in the database as an integer:

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

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

Array & JSON Casting

The `array` cast type is particularly useful when working with columns that are stored as serialized JSON. For example, if your database has a `JSON` or `TEXT` field type that contains serialized JSON, adding the `array` cast to that attribute will automatically deserialize the attribute to a PHP array when you access it on your Eloquent model:

```
<?php
```

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'options' => 'array',
    ];
}
```

Once the cast is defined, you may access the `options` attribute and it will automatically be deserialized from JSON into a PHP array. When you set the value of the `options` attribute, the given array will automatically be serialized back into JSON for storage:

```php
$user = App\User::find(1);

$options = $user->options;

$options['key'] = 'value';

$user->options = $options;

$user->save();
```

Date Casting

When using the `date` or `datetime` cast type, you may specify the date's format. This format will be used when the model is serialized to an array or JSON:

```php
/**
 * The attributes that should be cast to native types.
 *
 * @var array
 */
protected $casts = [
    'created_at' => 'datetime:Y-m-d',
];
```

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.