November 03, 2015

10 less-known (but awesome!) Laravel Collections methods

Laravel Eloquent is awesome - probably I don't need to tell you that. What is less known is the list of methods to work with Eloquent Collections. You can filter them, slice them, easily modify etc. But let's look at it one by one. A little notice here - that **Collections** are a broader term, whereas Eloquent Collections are the implementation of these broader Collections. So everything that we cover here in this article can be used for both instances. So, for example, you have Laravel code like this:

```
$books = Book::where('release_year', 2015)->get();
```

In this case **$books** is a collection, which is basically an array with additional features. For example it can look like this:

```
[
    ['title' => 'Lean Startup', 'price' => 10],
    ['title' => 'The One Thing', 'price' => 15],
    ['title' => 'Laravel: Code Bright', 'price' => 20],
    ['title' => 'The 4-Hour Work Week', 'price' => 5],
]
```

So what can we do with it? Important note: all those functions below are called without querying the database - so you run a query once and then perform all actions with collection "offline". **1. avg()** If you want to calculate average price of the book, you don't need to query database separately for that.

```
$average_price = $books->avg('price'); // in our case, 12.5
```

2. chunk() If you want to split the results by equal parts, for example to view them as columns, it can look like this:

```
$chunks = $books->chunk(2);
$chunks->toArray();
```

And the result:

```
[
[
    ['title' => 'Lean Startup', 'price' => 10],
    ['title' => 'The One Thing', 'price' => 15]
],
[
    ['title' => 'Laravel: Code Bright', 'price' => 20],
    ['title' => 'The 4-Hour Work Week', 'price' => 5],
]
]
```

3. contains() Simple – check whether our collection contains a certain value in one of the fields:

```
$books->contains('title', 'The One Thing'); // TRUE
$books->contains('title', 'The Second Thing'); // FALSE
```

4. every() Another method of splitting the collection into a different columns. This time – **every()** would form a new collection, consisting of every N-th element.

```
$books->every(2); // every 2nd element
```

Result:

```
[
    ['title' => 'Lean Startup', 'price' => 10],
    ['title' => 'Laravel: Code Bright', 'price' => 20],
]
```


5. filter() This method was already a topic of separate mini-article so will just briefly repeat:

```
$expensive_books = $books->filter(function ($book) {
    return $book->price > 10;
});
```

6. forget() This is a method to use if you want to get rid of one of the

columns.

```
$books->forget('price');
```

Result:

```
[
    ['title' => 'Lean Startup'],
    ['title' => 'The One Thing'],
    ['title' => 'Laravel: Code Bright'],
    ['title' => 'The 4-Hour Work Week'],
]
```

7. implode() Really similar to a well-known PHP array implode() function. The result is a joined string:

```
$books->implode('title', ', ');
```

Result:

```
'Lean Startup, The One Thing, Laravel: Code Bright, The 4-Hour Work Week'
```

8. keyBy() This is a really useful one to use with **foreach** loops later. Basically, it transforms a collection into an array with your chosen key.

```
$by_key = $books->keyBy('title');
$by_key->all();
```

Result:

```
[
    'Lean Startup' => ['title' => 'Lean Startup', 'price' => 10],
    'The One Thing' => ['title' => 'The One Thing', 'price' => 15],
    'Laravel: Code Bright' => ['title' => 'Laravel: Code Bright', 'price' => 20]
    'The 4-Hour Work Week' => ['title' => 'The 4-Hour Work Week', 'price' => 5],
]
```

9. map() If you want to extract some values and perform some actions with them – there's not only **foreach()**.

```
$discounted_books = $books->map(function ($item) {
    return ['title' => $item->title, 'price' => $item->price / 2];
});
```

Result:

```
[
    ['title' => 'Lean Startup', 'price' => 5],
    ['title' => 'The One Thing', 'price' => 7.5],
    ['title' => 'Laravel: Code Bright', 'price' => 10],
```

```
    ['title' => 'The 4-Hour Work Week', 'price' => 2.5],
]
```

10. pluck() This method allows you to extract just one column easily.

```
$prices = $books->pluck('price');
$plucked->all();
```

Result:

```
[10, 15, 20, 5]
```

So this is hand-picked 10 less-known methods, but it's not even a half of what's available. You can find full list with short examples in the official documentation for Eloquent Collections and Basic Collections. Keep digging deeper into Laravel world!

Written by

Povilas Korop

PHP web-developer with 20 years experience, working with Laravel since 2015. Over the years, published 1000+ videos on YouTube, dozens of full-length courses, hundreds of tutorials and thousands of tweets about Laravel.

Subscribe for 20+ new Laravel tutorials every week

E-mail address

You can unsubscribe at any time. You'll also get -20% off my courses!

YouTube Twitter GitHub