Prologue
Getting Started
Architecture Concepts
The Basics
Frontend
Security
Digging Deeper
Database
Eloquent ORM
Getting Started
Relationships
Collections
Mutators
API Resources
Serialization
Testing
Official Packages

Eloquent: Collections

Introduction
Available Methods
Custom Collections

Introduction

All multi-result sets returned by Eloquent are instances of the `Illuminate\Database\Eloquent\Collection` object, including results retrieved via the `get` method or accessed via a relationship. The Eloquent collection object extends the Laravel base collection, so it naturally inherits dozens of methods used to fluently work with the underlying array of Eloquent models.

All collections also serve as iterators, allowing you to loop over them as if they were simple PHP arrays:

```
$users = App\User::where('active', 1)->get();

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

However, collections are much more powerful than arrays and expose a variety of map / reduce operations that may be chained using an intuitive interface. For example, let's remove all inactive models and gather the first name for each remaining user:

```
$users = App\User::all();

$names = $users->reject(function ($user) {
    return $user->active === false;
})
->map(function ($user) {
    return $user->name;
});
```

While most Eloquent collection methods return a new instance of an Eloquent collection, the `pluck`, `keys`, `zip`, `collapse`, `flatten` and `flip` methods return a base collection instance. Likewise, if a `map` operation returns a collection that does not contain any Eloquent models, it will be automatically cast to a base collection.

Available Methods

All Eloquent collections extend the base Laravel collection object; therefore, they inherit all of the powerful methods provided by the base collection class.

In addition, the `Illuminate\Database\Eloquent\Collection` class provides a superset of methods to aid with managing your model collections. Most methods return `Illuminate\Database\Eloquent\Collection` instances; however, some methods return a base `Illuminate\Support\Collection` instance.

contains
diff
except
find
fresh
intersect
load
loadMissing
modelKeys
makeVisible
makeHidden
only
unique

`contains($key, $operator = null, $value = null)`

The `contains` method may be used to determine if a given model instance is contained by the collection. This method accepts a primary key or a model instance:

```
$users->contains(1);

$users->contains(User::find(1));
```

`diff($items)`

The `diff` method returns all of the models that are not present in the given collection:

```
use App\User;

$users = $users->diff(User::whereIn('id', [1, 2, 3])->get());
```

`except($keys)`

The `except` method returns all of the models that do not have the given primary keys:

```
$users = $users->except([1, 2, 3]);
```

`find($key)`

The `find` method finds a model that has a given primary key. If `$key` is a model instance, `find` will attempt to return a model matching the primary key. If `$key` is an array of keys, `find` will return all models which match the `$keys` using `whereIn()`:

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

$user = $users->find(1);
```

`fresh($with = [])`

The `fresh` method retrieves a fresh instance of each model in the collection from the database. In addition, any specified relationships will be eager loaded:

```
$users = $users->fresh();

$users = $users->fresh('comments');
```

`intersect($items)`

The `intersect` method returns all of the models that are also present in the given collection:

```
use App\User;

$users = $users->intersect(User::whereIn('id', [1, 2, 3])->get());
```

`load($relations)`

The `load` method eager loads the given relationships for all models in the collection:

```
$users->load('comments', 'posts');

$users->load('comments.author');
```

`loadMissing($relations)`

The `loadMissing` method eager loads the given relationships for all models in the collection if the relationships are not already loaded:

```
$users->loadMissing('comments', 'posts');

$users->loadMissing('comments.author');
```

`modelKeys()`

The `modelKeys` method returns the primary keys for all models in the collection:

```
$users->modelKeys();

// [1, 2, 3, 4, 5]
```

`makeVisible($attributes)`

The `makeVisible` method makes attributes visible that are typically "hidden" on each model in the collection:

```
$users = $users->makeVisible(['address', 'phone_number']);
```

`makeHidden($attributes)`

The `makeHidden` method hides attributes that are typically "visible" on each model in the collection:

```
$users = $users->makeHidden(['address', 'phone_number']);
```

`only($keys)`

The `only` method returns all of the models that have the given primary keys:

```
$users = $users->only([1, 2, 3]);
```

`unique($key = null, $strict = false)`

The `unique` method returns all of the unique models in the collection. Any models of the same type with the same primary key as another model in the collection are removed.

```
$users = $users->unique();
```

Custom Collections

If you need to use a custom `Collection` object with your own extension methods, you may override the `newCollection` method on your model:

```
<?php

namespace App;

use App\CustomCollection;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Create a new Eloquent Collection instance.
     *
     * @param  array  $models
     * @return \Illuminate\Database\Eloquent\Collection
     */
    public function newCollection(array $models = [])
    {
        return new CustomCollection($models);
    }
}
```

Once you have defined a `newCollection` method, you will receive an instance of your custom collection anytime Eloquent returns a `Collection` instance of that model. If you would like to use a custom collection for every model in your application, you should override the `newCollection` method on a base model class that is extended by all of your models.

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.