Course - Laravel Framework

Mã hóa Serialization

Khi xây dựng các API bằng Laravel, bạn thường cần biến đổi các model và mối quan hệ của mình sang mảng hoặc JSON.

Tags: serialization, ma hoa serialization, laravel

Giới thiệu

Khi xây dựng các API bằng Laravel, bạn thường cần biến đổi các model và mối quan hệ của mình sang mảng hoặc JSON. Eloquent bao gồm các phương pháp thuận tiện để thực hiện các chuyển đổi này, cũng như kiểm soát các thuộc tính attribute nào được mã hóa serialization của các model của bạn.

> Để có cách xử lý mã hóa serialize JSON model Eloquent và collection mạnh mẽ hơn nữa, hãy xem tài liệu về tài nguyên API của Eloquent.

Mã hóa serialize model và collection

Mã hóa serializing thành các mảng

Để chuyển đổi một model và các mối quan hệ đã tải về của nó thành một mảng, bạn nên sử dụng phương thức `toArray`. Phương thức này là đệ quy, vì vậy tất cả các thuộc tính attribute và tất cả các quan hệ (bao gồm cả các quan hệ của quan hệ) sẽ được chuyển đổi thành mảng:

```
use App\Models\User;

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

return $user->toArray();
```

Phương thức `attributesToArray` có thể được sử dụng để chuyển đổi các thuộc tính attribute của model thành một mảng chứ không phải các mối quan hệ của nó:

```
$user = User::first();

return $user->attributesToArray();
```

Bạn cũng có thể chuyển đổi toàn bộ collection các model thành mảng bằng cách gọi phương thức `toArray` trên đối tượng collection:

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

```php
return $users->toArray();
```

Mã hóa serialize thành JSON

Để chuyển đổi một model sang JSON, bạn nên sử dụng phương thức `toJson`. Giống như `toArray`, phương thức `toJson` là đệ quy, vì vậy tất cả các thuộc tính attribute và quan hệ sẽ được chuyển đổi thành JSON. Bạn cũng có thể chỉ định bất kỳ tùy chọn mã hóa JSON nào được PHP hỗ trợ:

```php
use App\Models\User;

$user = User::find(1);

return $user->toJson();

return $user->toJson(JSON_PRETTY_PRINT);
```

Ngoài ra, bạn có thể truyền một model hoặc collection vào một chuỗi, chuỗi này sẽ tự động gọi phương thức `toJson` trên model hoặc collection:

```php
return (string) User::find(1);
```

Vì các model và collection được chuyển đổi thành JSON khi truyền sang một chuỗi, bạn có thể trả về các đối tượng Eloquent trực tiếp từ các route hoặc controller của ứng dụng. Laravel sẽ tự động mã hóa serialize các model và collection Eloquent của bạn thành JSON khi chúng được trả về từ các route hoặc controller:

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

Các mối quan hệ

Khi một model Eloquent được chuyển đổi thành JSON, các mối quan hệ được tải của nó sẽ

tự động được đưa vào làm thuộc tính attribute trên đối tượng JSON. Ngoài ra, mặc dù các phương thức quan hệ Eloquent được khai báo bằng cách sử dụng cách đặt tên "camel case", thuộc tính JSON của mối quan hệ sẽ là "snake cake".

Ẩn các thuộc tính attribute từ JSON

Đôi khi bạn có thể muốn giới hạn các thuộc tính, chẳng hạn như mật khẩu, được bao gồm trong mảng hoặc dữ liệu JSON của model của bạn. Để làm như vậy, hãy thêm thuộc tính $hidden vào model của bạn. Trong các thuộc tính attribute được liệt kê trong mảng của thuộc tính $hidden sẽ không được đưa vào sữ liệu mã hóa serialize của model của bạn:

```php
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
  /**
   * The attributes that should be hidden for arrays.
   *
   * @var array
   */
  protected $hidden = ['password'];
}
```

> Để ẩn mối quan hệ, hãy thêm tên phương thức của mối quan hệ vào thuộc tính `$hidden` của mô hình Eloquent của bạn.

Ngoài ra, bạn có thể sử dụng thuộc tính `visible` để khai báo "danh sách cho phép" các thuộc tính attribute sẽ được bao gồm trong mảng và dữ liệu JSON của model của bạn. Tất cả các thuộc tính attribute không có trong mảng `$visible` sẽ bị ẩn đi khi model được chuyển đổi thành một mảng hoặc JSON:

```php
<?php

namespace App\Models;
```

```php
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
  /**
   * The attributes that should be visible in arrays.
   *
   * @var array
   */
  protected $visible = ['first_name', 'last_name'];
}
```

Tạm sửa đổi chế độ hiển thị thuộc tính

Nếu bạn muốn hiển thị một số thuộc tính attribute ẩn thường thấy trên một đối tượng model nào đó, bạn có thể sử dụng phương thức `makeVisible`. Phương thức `makeVisible` sẽ trả về đối tượng model:

```php
return $user->makeVisible('attribute')->toArray();
```

Tương tự như vậy, nếu bạn muốn ẩn một số thuộc tính attribute thường hiển thị, bạn có thể sử dụng phương thức `makeHidden`.

```php
return $user->makeHidden('attribute')->toArray();
```

Lắp thêm giá trị vào JSON

Đôi khi, khi chuyển đổi model thành mảng hoặc JSON, bạn có thể muốn thêm các thuộc tính attribute không có cột tương ứng trong cơ sở dữ liệu của mình. Để làm như vậy, trước tiên hãy khai báo một công cụ truy cập (accessor) cho giá trị:

```php
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Model;
```

```php
class User extends Model
{
  /**
   * Determine if the user is an administrator.
   *
   * @return bool
   */
  public function getIsAdminAttribute()
  {
    return $this->attributes['admin'] === 'yes';
  }
}
```

Sau khi tạo trình truy cập (accessor), thì hãy thêm tên thuộc tính attribute vào thuộc tính `appends` của model của bạn. Lưu ý rằng tên thuộc tính attribute thường được tham chiếu bằng cách sử dụng dữ liệu mã hóa serialize "snake case" của chúng, ngay cả khi phương thức PHP của trình truy cập (accessor) được khai báo bằng "camel case":

```php
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class User extends Model
{
  /**
   * The accessors to append to the model's array form.
   *
   * @var array
   */
  protected $appends = ['is_admin'];
}
```

Khi thuộc tính đã được thêm vào danh sách `appends`, nó sẽ được đưa vào cả mảng và các dữ liệu JSON của model. Các thuộc tính attribute trong mảng `appends` cũng sẽ có các cài đặt `visible` và `hidden` được cấu hình trên model.

Lắp thêm ngay khi đang chạy

Trong thời gian chạy, bạn có thể hướng dẫn một đối tượng model lắp thêm các thuộc tính attribute bổ sung bằng cách sử dụng phương thức `append`. Hoặc, bạn có thể sử dụng phương thức `setAppends` để ghi đè toàn bộ mảng thuộc tính được lắp thêm cho một đối tượng model nào đó:

```
return $user->append('is_admin')->toArray();

return $user->setAppends(['is_admin'])->toArray();
```

Mã hóa serialization ngày tháng

Tùy chỉnh định dạng ngày mặc định

Bạn có thể tùy chỉnh định dạng mã hóa serialize mặc định bằng cách ghi đè phương thức `serializeDate`. Phương thức này không ảnh hưởng đến cách định dạng ngày tháng của bạn khi lưu trữ trong cơ sở dữ liệu:

```
/**
 * Prepare a date for array / JSON serialization.
 *
 * @param  \DateTimeInterface  $date
 * @return string
 */
protected function serializeDate(DateTimeInterface $date)
{
  return $date->format('Y-m-d');
}
```

Tùy chỉnh định dạng ngày cho mỗi thuộc tính

Bạn có thể tùy chỉnh định dạng mã hóa serialize của từng thuộc tính Eloquent attribute liên quan ngày tháng bằng cách mô tả định dạng ngày tháng trong khai báo chức năng đổi kiểu của model:

```php
protected $casts = [
  'birthday' => 'date:Y-m-d',
  'joined_at' => 'datetime:Y-m-d H:00',
];
```