Course - Laravel Framework

Mutator/Cast

Truy cập, biến đổi và chuyển đổi kiểu thuộc tính cho phép bạn chuyển đổi các giá trị thuộc tính attribute của Eloquent khi bạn truy xuất hoặc đặt chúng trên các đối tượng của model.

Tags: bien doi, chuyen doi, Mutator/Cast, laravel

Giới thiệu

Truy cập, biến đổi và chuyển đổi kiểu thuộc tính cho phép bạn chuyển đổi các giá trị thuộc tính attribute của Eloquent khi bạn truy xuất hoặc đặt chúng trên các đối tượng của model. Ví dụ: bạn có thể muốn sử dụng tính năng mã hóa Laravel encrypter để mã hóa một giá trị trong khi nó được lưu trữ trong cơ sở dữ liệu, sau đó tự động giải mã thuộc tính attribute khi bạn truy cập nó trên Eloquent model. Hoặc, bạn có thể muốn chuyển đổi một chuỗi JSON được lưu trữ trong cơ sở dữ liệu của bạn thành một mảng khi nó được truy cập thông qua Eloquent model của bạn.

Truy cập và biến đổi

Khai báo một trình truy cập (accessor)

Trình truy cập sẽ chuyển đổi giá trị thuộc tính attribute Eloquent khi nó được truy cập. Để khai báo một accessor, hãy tạo một phương thức `get{Attribute}Attribute` trên model của bạn trong đó `{Attribute}` là tên viết hoa "studly" của cột mà bạn muốn truy cập.

Trong ví dụ này, chúng ta sẽ khai báo một accessor cho thuộc tính `first_name`. accessor sẽ tự động được gọi bởi Eloquent khi cố gắng truy xuất giá trị của thuộc tính `first_name`:

```php
<?php

namespace App\Models;

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);
```

```
    }
}
```

Như bạn có thể thấy, giá trị ban đầu của cột được truyền cho accessor, cho phép bạn thao tác và trả về giá trị. Để truy cập giá trị của accessor, bạn có thể chỉ cần truy cập thuộc tính `first_name` trên một đối tượng model:

```
use App\Models\User;

$user = User::find(1);

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

Bạn không bị giới hạn trong việc tương tác với một thuộc tính attribute trong accessor của mình. Bạn cũng có thể sử dụng accessor để trả về các giá trị mới, được tính toán từ các thuộc tính attribute hiện có:

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

> Nếu bạn muốn các giá trị được tính toán này được thêm vào những sự thể hiện mảng/JSON của model của mình, bạn sẽ cần phải nối chúng.

Khai báo một trình biến đổi

Một trình biến (*munator*) đổi sẽ biến đổi một giá trị thuộc tính attribute Eloquent khi nó được thiết lập. Để khai báo một munator, hãy khai báo một phương thức `set{Attribute}Attribute` trên model của bạn trong đó `{Attribute}` là tên được viết bằng chữ "đặc biệt" của cột mà bạn muốn truy cập.

Hãy khai báo một mutator cho thuộc tính `first_name`. Mutator (trình biến đổi) này sẽ tự động được gọi khi chúng ta cố gắng đặt giá trị của thuộc tính attribute `first_name` trên model:

```php
<?php

namespace App\Models;
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);
  }
}
```

Trình biến đổi (*munator*) sẽ nhận giá trị đang được cài đặt trên thuộc tính attribute, cho phép bạn thao tác giá trị và đặt giá trị được thao tác trên thuộc tính `$attributes` nội bộ của model Eloquent. Để sử dụng trình biến đổi của chúng ta, chúng ta chỉ cần cài đặt thuộc tính `first_name` trên model Eloquent:

```php
use App\Models\User;

$user = User::find(1);

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

Trong ví dụ này, hàm `setFirstNameAttribute` sẽ được gọi với giá trị `Sally`. Sau đó, trình biến đổi sẽ áp dụng hàm `strtolower` cho tên và cài đặt giá trị kết quả của nó trong mảng thuộc tính `$attributes` nội bộ.

Chuyển đổi kiểu thuộc tính attribute

Đổi kiểu thuộc tính attribute cung cấp chức năng tương tự như accessor và mutator mà không yêu cầu bạn khai báo bất kỳ phương thức bổ sung nào trên model của mình. Thay vào đó, thuộc tính `$casts` của model của bạn cung cấp một phương pháp thuận tiện để chuyển đổi các thuộc tính attribute thành các kiểu dữ liệu thông thường.

Thuộc tính `$casts` phải là một mảng trong đó khóa là tên của thuộc tính attribute đang được đổi kiểu và giá trị là kiểu bạn muốn đổi. Các loại kiểu có thể chuyển đổi được là:

- `array`
- `AsStringable::class`
- `boolean`
- `collection`
- `date`
- `datetime`
- `immutable_date`
- `immutable_datetime`
- `decimal:<digits>`
- `double`
- `encrypted`
- `encrypted:array`
- `encrypted:collection`
- `encrypted:object`
- `float`
- `integer`
- `object`
- `real`
- `string`
- `timestamp`

Để chứng minh việc đổi kiểu thuộc tính attribute, hãy đổi kiểu của thuộc tính `is_admin`, được lưu trữ trong cơ sở dữ liệu của chúng tôi dưới dạng số nguyên (`0` hoặc `1`) thành giá trị boolean:

```php
<?php

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

class User extends Model
```

```
{
  /**
   * The attributes that should be cast.
   *
   * @var array
   */
  protected $casts = [
    'is_admin' => 'boolean',
  ];
}
```

Sau khi đổi kiểu, thuộc tính attribute `is_admin` sẽ luôn được chuyển thành *boolean* khi bạn truy cập nó, ngay cả khi giá trị cơ bản được lưu trữ trong cơ sở dữ liệu dưới dạng số nguyên:

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

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

Nếu bạn cần thêm một lần đổi kiểu mới, tạm thời trong thời gian chạy, bạn có thể sử dụng phương thức `mergeCasts`. Các khai báo việc đổi kiểu này sẽ được thêm vào bất kỳ sự đổi kiểu nào đã được khai báo trên model:

```
$user->mergeCasts([
  'is_admin' => 'integer',
  'options' => 'object',
]);
```

> **Chú ý:** Các thuộc tính attribute không có giá trị (null) sẽ không đổi kiểu được. Ngoài ra, bạn không bao giờ nên khai báo việc đổi kiểu (hoặc một thuộc tính) có cùng tên với một mối quan hệ.

Chuyển đổi kiểu chuỗi câu

Bạn có thể sử dụng class đổi kiểu như `Illuminate\Database\Eloquent\Casts\AsStringable` để đổi kiểu một thuộc tính attribute của model cho một đối tượng `Illuminate\Support\Stringable`:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Casts\AsStringable;
use Illuminate\Database\Eloquent\Model;

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

Đổi kiểu với Mảng và JSON

Việc đổi kiểu `array` cực kỳ hữu ích khi làm việc với các cột được lưu trữ dưới dạng JSON được mã hóa (serialized). Ví dụ: nếu cơ sở dữ liệu của bạn có loại trường `JSON` hoặc `TEXT` có chứa JSON được mã hóa (serialized), việc thêm việc đổi kiểu `array` vào thuộc tính attribute đó sẽ tự động giải mã hóa thuộc tính attribute thành một mảng PHP khi bạn truy cập nó trên model Eloquent của mình:

```php
<?php

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

class User extends Model
```

```
{
  /**
   * The attributes that should be cast.
   *
   * @var array
   */
  protected $casts = [
    'options' => 'array',
  ];
}
```

Sau khi việc đổi kiểu đã được khai báo, thì bạn có thể truy cập thuộc tính attribute `options` và nó sẽ tự động được giải mã từ JSON thành một mảng PHP. Khi bạn đặt giá trị của thuộc tính attribute `options`, mảng đã cho sẽ tự động được mã hóa trở lại thành JSON để lưu trữ:

```
use App\Models\User;

$user = User::find(1);

$options = $user->options;

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

$user->options = $options;

$user->save();
```

Để cập nhật một trường đơn lẻ của thuộc tính attribute JSON với cú pháp ngắn gọn hơn, bạn có thể sử dụng toán tử `->` khi gọi phương thức `update`:

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

$user->update(['options->key' => 'value']);
```

Đổi kiểu đối tượng mảng và collection

Mặc dù việc đổi kiểu mảng tiêu chuẩn là đủ cho nhiều ứng dụng, nhưng nó có một số nhược điểm. Vì mảng đổi kiểu sẽ trả về kiểu nguyên thủy *primitive type*, nên không thể thay đổi trực tiếp offset nào đó của mảng. Ví dụ: đoạn mã sau sẽ gây ra lỗi PHP:

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

$user->options['key'] = $value;
```

Để giải quyết vấn đề này, Laravel cung cấp một bản đổi kiểu `AsArrayObject` mà sẽ đổi kiểu thuộc tính attribute JSON của bạn thành một class `ArrayObject`. Tính năng này được thực hiện bằng cách triển khai đổi kiểu tùy chỉnh của Laravel, cho phép Laravel lưu vào bộ nhớ cache và biến đổi đối tượng bị biến đổi một cách thông minh để các offset riêng lẻ có thể được sửa đổi mà không gây ra lỗi PHP. Để sử dụng bản đổi kiểu `AsArrayObject`, chỉ cần gán nó cho một thuộc tính attribute:

```
use Illuminate\Database\Eloquent\Casts\AsArrayObject;

/**
 * The attributes that should be cast.
 *
 * @var array
 */
protected $casts = [
  'options' => AsArrayObject::class,
];
```

Tương tự như vậy, Laravel cung cấp một kiểu đổi kiểu `AsCollection` sẽ đổi kiểu thuộc tính attribute JSON của bạn thành một đối tượng Laravel Collection:

```
use Illuminate\Database\Eloquent\Casts\AsCollection;

/**
 * The attributes that should be cast.
 *
 * @var array
 */
protected $casts = [
```

```
    'options' => AsCollection::class,
];
```

Đổi kiểu ngày tháng (Date)

Theo mặc định, Eloquent sẽ chuyển đổi kiểu các cột `created_at` và `updated_at` thành các đối tượng Carbon, mở rộng class PHP `DateTime` và cung cấp một loạt các phương thức ngày tháng hữu ích. Bạn có thể đổi kiểu các thuộc tính attribute ngày tháng bổ sung bằng cách khai báo các lần đổi kiểu ngày tháng bổ sung trong thuộc tính mảng `$casts` của model của bạn. Thông thường, ngày tháng nên được chuyển đổi bằng cách sử dụng kiểu `datetime` hoặc `immutable_datetime`.

Khi xác định chuyển đổi `date` hoặc `datetime`, bạn cũng có thể chỉ định định dạng của ngày tháng đó. Định dạng này sẽ được sử dụng khi model được mã hóa serialized thành một mảng hoặc JSON:

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

Khi một cột được đổi kiểu dạng ngày, bạn có thể đặt giá trị thuộc tính attribute model tương ứng thành dấu thời gian UNIX, chuỗi ngày (`Y-m-d`), chuỗi ngày-giờ hoặc đối tượng `DateTime`/`Carbon`. Giá trị của ngày tháng sẽ được chuyển đổi chính xác và được lưu trữ trong cơ sở dữ liệu của bạn.

Bạn có thể tùy chỉnh định dạng mã hóa mặc định cho tất cả các ngày của model bằng cách khai báo phương thức `serializeDate` trên model của bạn. 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');
    }
```

Để chỉ định định dạng sẽ được sử dụng khi thực sự lưu trữ ngày tháng của một model trong cơ sở dữ liệu của bạn, bạn nên xác định thuộc tính `$dateFormat` trên model của mình:

```
    /**
     * The storage format of the model's date columns.
     *
     * @var string
     */
    protected $dateFormat = 'U';
```

Kiểu ngày tháng, mã hóa và timezone

Theo mặc định, các lần đổi kiểu ngày và giờ sẽ mã hóa ngày thành chuỗi ngày UTC ISO-8601 (`1986-05-28T21: 05: 54.000000Z`), bất kể múi giờ được chỉ định trong tùy chọn cấu hình `timezone` của ứng dụng của bạn. Chúng tôi đặc biệt khuyến khích bạn luôn sử dụng định dạng mã hóa serialize này, cũng như lưu trữ ngày tháng của ứng dụng trong múi giờ UTC bằng cách không thay đổi tùy chọn cấu hình `timezone` của ứng dụng từ giá trị `UTC` mặc định của nó. Việc sử dụng nhất quán múi giờ UTC trong toàn bộ ứng dụng của bạn sẽ cung cấp mức độ tương tác tối đa với các thư viện thao tác ngày tháng khác được viết bằng PHP và JavaScript.

Nếu áp dụng định dạng tùy chỉnh cho việc đổi kiểu date hoặc datetime, chẳng hạn như `datetime:Y-m-d H:i:s`, múi giờ bên trong của đối tượng Carbon sẽ được sử dụng trong quá trình mã hóa ngày. Thông thường, đây sẽ là múi giờ được chỉ định trong tùy chọn cấu hình `timezone` của ứng dụng của bạn.

Đổi kiểu enum

> **Chú ý:** Đổi kiểu chỉ có trong PHP 8.1 trở lên

Eloquent cũng cho phép bạn truyền các giá trị thuộc tính attribute của mình sang các enum PHP. Để thực hiện điều này, bạn có thể chỉ định thuộc tính attribute và enum bạn muốn đổi kiểu trong mảng thuộc tính `$casts` của model của bạn:

```
use App\Enums\ServerStatus;

/**
 * The attributes that should be cast.
 *
 * @var array
 */
protected $casts = [
  'status' => ServerStatus::class,
];
```

Khi bạn đã khai báo kiểu sẽ đổi trên model của mình, thuộc tính attribute được mô tả sẽ tự động được chuyển đổi thành và từ một enum khi bạn tương tác với thuộc tính:

```
if ($server->status == ServerStatus::provisioned) {
  $server->status = ServerStatus::ready;

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

Đổi kiểu Encrypted

Đổi kiểu `encrypted` sẽ mã hóa một giá trị thuộc tính attribute của model bằng cách sử dụng các tính năng mã hóa có sẵn của Laravel. Ngoài ra, đổi kiểu theo `encrypted:array`, `encrypted:collection`, `encrypted:object`, `AsEncryptedArrayObject` và `AsEncryptedCollection` sẽ làm việc như các đối tác không được mã hóa của chúng. Tuy nhiên, như bạn mong chờ, giá trị căn bản sẽ được mã hõa khi được lưu trữ trong cơ sở dữ liệu.

Vì không thể đoán trước được độ dài cuối cùng của văn bản được mã hóa và dài hơn đối với văn bản thuần túy của nó, nên hãy đảm bảo cột cơ sở dữ liệu đã được liên kết là loại

`TEXT` hoặc lớn hơn. Ngoài ra, vì các giá trị được mã hóa trong cơ sở dữ liệu, nên bạn sẽ không thể truy vấn hoặc tìm kiếm các giá trị thuộc tính được mã hóa.

Chuyển đổi thời gian truy vấn

Đôi khi bạn có thể cần phải áp dụng chuyển đổi kiểu trong khi thực hiện truy vấn, chẳng hạn như khi chọn giá trị thô từ bảng. Ví dụ: hãy xem xét truy vấn sau:

```
use App\Models\Post;
use App\Models\User;

$users = User::select([
  'users.*',
  'last_posted_at' => Post::selectRaw('MAX(created_at)')
    ->whereColumn('user_id', 'users.id')
])->get();
```

Thuộc tính attribute `last_posted_at` trên kết quả của truy vấn này sẽ là một chuỗi đơn giản. Sẽ thật tuyệt vời nếu chúng ta có thể áp dụng ngày giờ cho thuộc tính attribute này khi thực hiện truy vấn. Rất may, chúng ta có thể thực hiện điều này bằng cách sử dụng phương thức `withCasts`:

```
$users = User::select([
  'users.*',
  'last_posted_at' => Post::selectRaw('MAX(created_at)')
    ->whereColumn('user_id', 'users.id')
])->withCasts([
  'last_posted_at' => 'datetime'
])->get();
```

Tự tạo kiểu chuyển đổi

Laravel có nhiều kiểu cast tích hợp, hữu ích; tuy nhiên, đôi khi bạn có thể cần phải tự tạo kiểu đổi của riêng mình. Bạn có thể thực hiện điều này bằng cách mô tả một class thực thi interface `CastsAttributes`.

Các class thực thi interface này phải mô tả phương thức `get` và `set`. Phương thức `get` chịu

trách nhiệm chuyển đổi giá trị thô từ cơ sở dữ liệu thành giá trị kiểu đổi, trong khi phương thức `set` sẽ chuyển đổi giá trị kiểu đổi thành giá trị thô có thể được lưu trữ trong cơ sở dữ liệu. Ví dụ: chúng ta sẽ thực thi lại việc đổi kiểu kiểu `json` được tích hợp sẵn dưới dạng kiểu *đổi kiểu tự tạo*:

```php
<?php

namespace App\Casts;
use Illuminate\Contracts\Database\Eloquent\CastsAttributes;

class Json implements CastsAttributes
{
  /**
   * Cast the given value.
   *
   * @param  \Illuminate\Database\Eloquent\Model  $model
   * @param  string  $key
   * @param  mixed  $value
   * @param  array  $attributes
   * @return array
   */
  public function get($model, $key, $value, $attributes)
  {
    return json_decode($value, true);
  }

  /**
   * Prepare the given value for storage.
   *
   * @param  \Illuminate\Database\Eloquent\Model  $model
   * @param  string  $key
   * @param  array  $value
   * @param  array  $attributes
   * @return string
   */
  public function set($model, $key, $value, $attributes)
  {
```

```
        return json_encode($value);
    }
}
```

Khi bạn đã tạo *đổi kiểu tự tạo*, bạn có thể đính kèm nó vào thuộc tính attribute của model bằng cách sử dụng tên class của nó:

```
<?php

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

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

Đổi kiểu đối tượng giá trị

Bạn không bị giới hạn trong việc đổi kiểu giá trị cho các kiểu dữ liệu thuần. Bạn cũng có thể đổi kiểu giá trị cho các đối tượng. Việc khai báo kiểu đổi tự tạo nhằm đổi kiểu giá trị cho các đối tượng rất giống với việc đổi kiểu cho các kiểu dữ liệu thuần; tuy nhiên, phương thức `set` sẽ trả về một mảng các cặp khóa/giá trị sẽ được sử dụng để đặt các giá trị thô, lưu trữ trên model.

Ví dụ, chúng tôi sẽ khai báo một class đổi kiểu tự tạo nhằm đổi kiểu nhiều giá trị model vào một đối tượng giá trị `Address` đơn nhất. Chúng tôi sẽ giả định giá trị `Address` có hai thuộc tính public: `lineOne` và `lineTwo`:

```php
<?php

namespace App\Casts;

use App\Models\Address as AddressModel;
use Illuminate\Contracts\Database\Eloquent\CastsAttributes;
use InvalidArgumentException;

class Address implements CastsAttributes
{
  /**
   * Cast the given value.
   *
   * @param  \Illuminate\Database\Eloquent\Model  $model
   * @param  string  $key
   * @param  mixed  $value
   * @param  array  $attributes
   * @return \App\Models\Address
   */
  public function get($model, $key, $value, $attributes)
  {
    return new AddressModel(
      $attributes['address_line_one'],
      $attributes['address_line_two']
    );
  }

  /**
   * Prepare the given value for storage.
   *
   * @param  \Illuminate\Database\Eloquent\Model  $model
   * @param  string  $key
   * @param  \App\Models\Address  $value
   * @param  array  $attributes
   * @return array
   */
  public function set($model, $key, $value, $attributes)
```

```
    {
        if (! $value instanceof AddressModel) {
            throw new InvalidArgumentException('The given value is not an Address instance.');
        }

        return [
            'address_line_one' => $value->lineOne,
            'address_line_two' => $value->lineTwo,
        ];
    }
}
```

Khi đổi kiểu các đối tượng giá trị, mọi thay đổi được thực hiện đối với đối tượng giá trị sẽ tự động được đồng bộ hóa trở lại model trước khi model được lưu:

```
use App\Models\User;

$user = User::find(1);

$user->address->lineOne = 'Updated Address Value';

$user->save();
```

> Nếu bạn định mã hóa các model Eloquent của mình chứa các đối tượng giá trị thành JSON hoặc mảng, bạn nên thực thi các interface `Illuminate\Contracts\Support\Arrayable` và `JsonSerializable` trên đối tượng giá trị.

Mã hóa serialization Array/JSON

Khi một model Eloquent được chuyển đổi thành một mảng hoặc JSON bằng cách sử dụng các phương thức toArray và toJson, các đối tượng giá trị đổi kiểu tự tạo của bạn thường sẽ được mã hóa miễn là chúng thực thi các interface `Illuminate\Contracts\Support\Arrayable` và `JsonSerializable`. Tuy nhiên, khi sử dụng các đối tượng giá trị do thư viện bên thứ ba cung cấp, bạn có thể không có khả năng thêm các giao diện này vào đối tượng.

Do đó, bạn có thể chỉ định rằng class đổi kiểu tự tạo của bạn sẽ chịu trách nhiệm mã hóa

serialize đối tượng giá trị. Để làm như vậy, class đổi kiểu tự tạo của bạn phải thực thi interface `Illuminate\Contracts\Database\Eloquent\SerializesCastableAttributes`. Interface này nói rằng class của bạn phải chứa một phương thức mã hóa serialize sẽ trả về dạng mã hóa serialize của đối tượng giá trị của bạn:

```
/**
 * Get the serialized representation of the value.
 *
 * @param  \Illuminate\Database\Eloquent\Model  $model
 * @param  string  $key
 * @param  mixed  $value
 * @param  array  $attributes
 * @return mixed
 */
public function serialize($model, string $key, $value, array $attributes)
{
  return (string) $value;
}
```

Chuyển đổi kiểu Inbound

Đôi khi, bạn có thể cần phải viết một số cách đổi kiểu tùy chỉnh chỉ để chuyển đổi các giá trị đang được đặt trên model và không thực hiện bất kỳ hoạt động nào khi các thuộc tính attribute đang được truy xuất từ model. Một ví dụ cổ điển về việc đổi kiểu inbound là "hasing" cast. Chuyển đổi kiểu inbound tùy chỉnh nên thực thi interface `CastsInboundAttributes`, giao diện này chỉ yêu cầu khai báo phương thức `set`.

```
<?php

namespace App\Casts;
use Illuminate\Contracts\Database\Eloquent\CastsInboundAttributes;

class Hash implements CastsInboundAttributes
{
  /**
   * The hashing algorithm.
   *
```

```
     * @var string
     */
    protected $algorithm;

    /**
     * Create a new cast class instance.
     *
     * @param  string|null  $algorithm
     * @return void
     */
    public function __construct($algorithm = null)
    {
      $this->algorithm = $algorithm;
    }

    /**
     * Prepare the given value for storage.
     *
     * @param  \Illuminate\Database\Eloquent\Model  $model
     * @param  string  $key
     * @param  array  $value
     * @param  array  $attributes
     * @return string
     */
    public function set($model, $key, $value, $attributes)
    {
      return is_null($this->algorithm)
        ? bcrypt($value)
        : hash($this->algorithm, $value);
    }
}
```

Các tham số đổi kiểu

Khi đính kèm một khai báo đổi kiểu tự tạo vào một model, các tham số đổi kiểu có thể được chỉ định bằng cách tách chúng khỏi tên class bằng cách sử dụng ký tự `::` và phân tách nhiều tham số bằng dấu phẩy. Các tham số sẽ được chuyển cho hàm tạo của class đổi kiểu:

```php
/**
 * The attributes that should be cast.
 *
 * @var array
 */
protected $casts = [
  'secret' => Hash::class.':sha256',
];
```

Có thể đổi kiểu được

Bạn có thể muốn cho phép các đối tượng giá trị của ứng dụng xác định các class đổi kiểu tự tạo của riêng chúng. Thay vì đính kèm class đổi kiểu tùy chỉnh vào model của bạn, bạn có thể đính kèm theo cách khác một class đối tượng giá trị thực thi interface `Illuminate\Contracts\Database\Eloquent\Castable`:

```php
use App\Models\Address;

protected $casts = [
  'address' => Address::class,
];
```

Các đối tượng thực thi interface `Castable` phải khai báo phương thức `castUsing` sẽ trả về tên class của class đổi kiểu tự tạo mà chịu trách nhiệm chuyển đổi kiểu qua và trở lại class `Castable`:

```php
<?php

namespace App\Models;

use Illuminate\Contracts\Database\Eloquent\Castable;
use App\Casts\Address as AddressCast;

class Address implements Castable
{
  /**
```

```
     * Get the name of the caster class to use when casting from / to this cast target.
     *
     * @param  array  $arguments
     * @return string
     */
    public static function castUsing(array $arguments)
    {
        return AddressCast::class;
    }
}
```

Khi sử dụng các class `Castable`, bạn vẫn có thể cung cấp các đối số trong khai báo `$casts`. Các đối số sẽ được truyền đến phương thức `castUsing`.

```
use App\Models\Address;

protected $casts = [
    'address' => Address::class.':argument',
];
```

Castable & lớp cast ẩn danh

Bằng cách kết hợp "castable" với các class ẩn danh của PHP, bạn có thể khai báo một đối tượng giá trị và logic đổi kiểu của nó như một đối tượng có thể đổi kiểu đơn lẻ. Để thực hiện điều này, hãy trả về một class ẩn danh từ phương thức `castUsing` của đối tượng giá trị của bạn. Lớp ẩn danh sẽ thực thi interface `CastsAttributes`:

```
<?php

namespace App\Models;

use Illuminate\Contracts\Database\Eloquent\Castable;
use Illuminate\Contracts\Database\Eloquent\CastsAttributes;

class Address implements Castable
{
```

```php
    // ...

    /**
     * Get the caster class to use when casting from / to this cast target.
     *
     * @param  array  $arguments
     * @return object|string
     */
    public static function castUsing(array $arguments)
    {
        return new class implements CastsAttributes
        {
            public function get($model, $key, $value, $attributes)
            {
                return new Address(
                    $attributes['address_line_one'],
                    $attributes['address_line_two']
                );
            }

            public function set($model, $key, $value, $attributes)
            {
                return [
                    'address_line_one' => $value->lineOne,
                    'address_line_two' => $value->lineTwo,
                ];
            }
        };
    }
}
```