Refactoring to Collections

or why Collections are g0il ❤

karriere.at

About me

- Web Developer since 2006
- currently working @ karriere.at
- actual project: next version of www.karriere.at
- Laravel Fanboy ❤

karriere.at

- biggest job platform in Austria (100k unique clients per day)
- ~ 130 employees
- ~ 30 developers

Overview

- Collections wtf?
- From loops ... to array_* ... to Collections
- array_* functions in PHP
- The empty/null problem
- Performance
- Examples
- Using Collections

karriere.at

Collections wtf?

- arrays are cool
- but you often write the same code over and over again
- collections provide high order functions (`map, filter, reduce, ...`) inspired by functional programming
- wrapper for `array_*` functions
- force pipeline style programming

karriere.at

From loops ...

```
private function getEmailsFromActiveUsers($users)
{









  // returns john.doe@karriere.at,yada.yada@karriere.at
}
```

From loops ...

```php
private function getEmailsFromActiveUsers($users)
{


  foreach ($users as $user) {



  }


}
```

karriere.at

From loops ...

```
private function getEmailsFromActiveUsers($users)
{


  foreach ($users as $user) {
    if ($user->isActive) {

    }
  }


}
```

From loops ...

```php
private function getEmailsFromActiveUsers($users)
{
  $emailAddresses = [];

  foreach ($users as $user) {
    if ($user->isActive) {
      $emailAddresses[] = $user->email;
    }
  }


}
```

From loops ...

```php
private function getEmailsFromActiveUsers($users)
{
  $emailAddresses = [];

  foreach ($users as $user) {
    if ($user->isActive) {
      $emailAddresses[] = $user->email;
    }
  }

  return implode(',', $emailAddresses);
}
```

… over array_* …

```
private function getEmailsFromActiveUsers($users)
{









}
```

… over array_* …

```
private function getEmailsFromActiveUsers($users)
{
  $activeUsers = array_filter($users, function ($user) {
    return $user->isActive;
  });







}
```

karriere.at

… over array_* …

```
private function getEmailsFromActiveUsers($users)
{
  $activeUsers = array_filter($users, function ($user) {
    return $user->isActive;
  });

  $emailAddresses = array_map(function ($user) {
    return $user->email;
  }, $activeUsers);


}
```

… over array_* …

```
private function getEmailsFromActiveUsers($users)
{
  $activeUsers = array_filter($users, function ($user) {
    return $user->isActive;
  });

  $emailAddresses = array_map(function ($user) {
    return $user->email;
  }, $activeUsers);

  return implode(',', $emailAddresses);
}
```

… over array_* (chained) …

```php
private function getEmailsFromActiveUsers($users)
{
  return implode(
    ',',
    array_map(function ($user) {
        return $user->email;
      },
      array_filter(
        $users,
        function ($user) {
          return $user->isActive;
        }
      )
    )
  );
}
```

karriere.at

… over array_* (chained) …

```
private function getEmailsFromActiveUsers($users)
{
  return implode(
    ',',
    array_map(function ($user) {
        return $user->email;
      },
      array_filter(
        $users,
        function ($user) {
          return $user->isActive;
        }
      )
    )
  );
}
```

looks weird and is difficult to understand

… to Collections

```
private function getEmailsFromActiveUsers($users)
{
  return collect($users)









}
```

… to Collections

```
private function getEmailsFromActiveUsers($users)
{
  return collect($users)
    ->filter(function ($user) {
      return $user->isActive;
    })




}
```

... to Collections

```php
private function getEmailsFromActiveUsers($users)
{
  return collect($users)
    ->filter(function ($user) {
      return $user->isActive;
    })
    ->map(function ($user) {
      return $user->email;
    })

}
```

karriere.at

... to Collections

```
private function getEmailsFromActiveUsers($users)
{
  return collect($users)
    ->filter(function ($user) {
      return $user->isActive;
    })
    ->map(function ($user) {
      return $user->email;
    })
    ->implode(',');
}
```

... to Collections

```
private function getEmailsFromActiveUsers($users)
{
  return collect($users)
    ->where('isActive', true)
    ->map(function ($user) {
      return $user->email;
    })
    ->implode(',');
}
```

replaced `filter` with `where`

karriere.at

... to Collections

```
private function getEmailsFromActiveUsers($users)
{
  return collect($users)
    ->where('isActive', true)
    ->implode('email', ',');
}
```

the `implode` method can take 2 parameters

1. the field to concatinate
2. the glue

array_* functions in PHP

- have an odd parameter order
  - `array_map(callable $callback, array $array1 [, array $... ])`
  - `array_filter(array $array [, callable $callback [, int $flag = 0 ]])`
- no support for pipeline calls
- features like `where`, `pluck`, `groupBy`, … missing
- **but they are used by Collections**

karriere.at

The empty/null problem

- retrieving data from 3rd party libraries
- the method returns an `array` but sometimes `null`

The empty/null problem

- retrieving data from 3rd party libraries
- the method returns an `array` but sometimes `null`

```
$users = $repository->getUsers();

if( !is_null($users) && count($users) > 0) {
  // do something
}
```

The empty/null problem

- retrieving data from 3rd party libraries
- the method returns an `array` but sometimes `null`

```php
$users = $repository->getUsers();

if( !is_null($users) && count($users) > 0) {
  // do something
}
```

or

```php
$users = $repository->getUsers();

if( !empty($users)) {
  // do something
}
```

… solved by Collections

```
$users = collect($repository->getUsers());
```

… solved by Collections

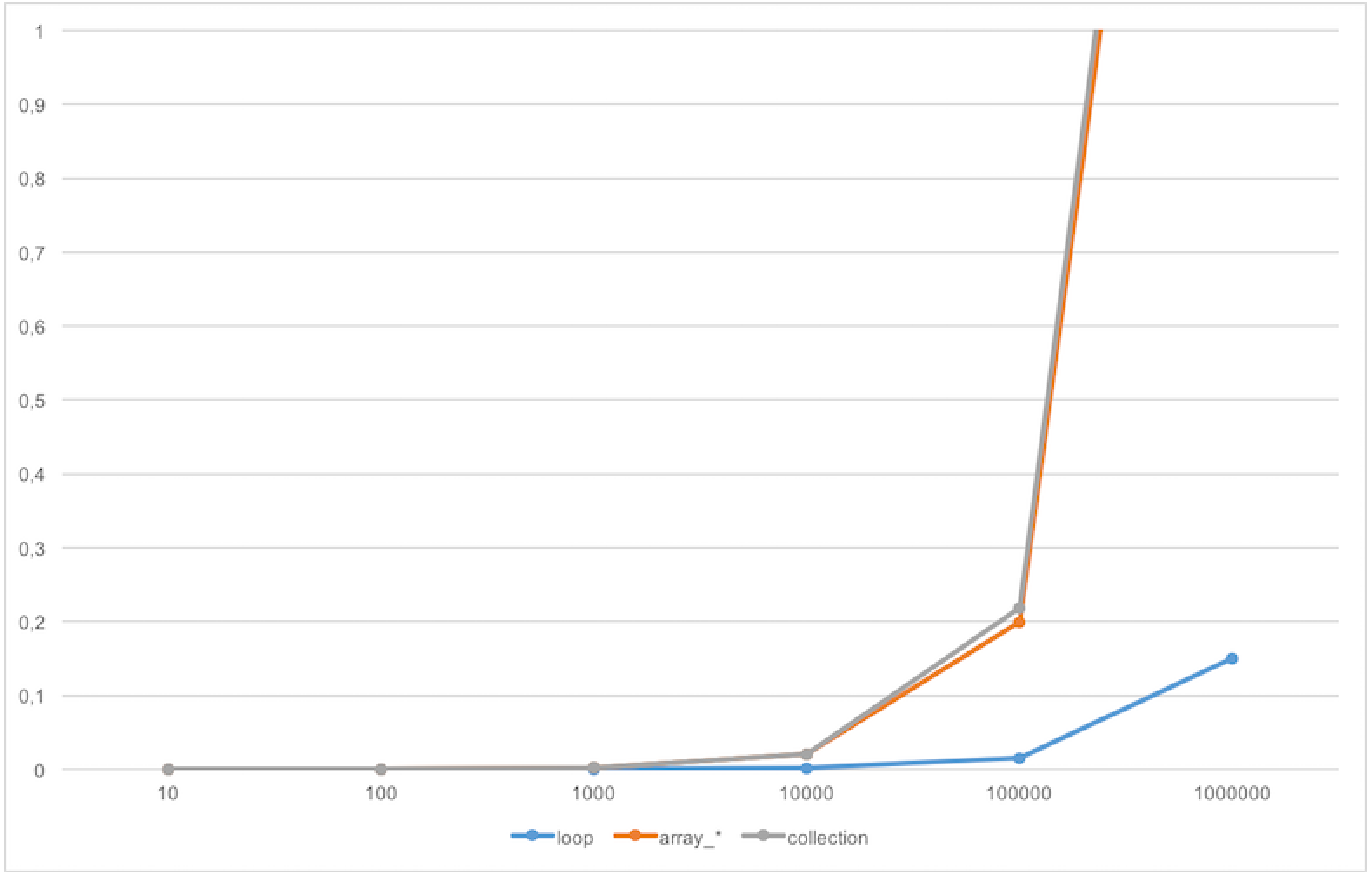
```
$users = collect($repository->getUsers());
```

	resulting array	isEmpty
`collect(null)`	`[]`	`true`
`collect([])`	`[]`	`true`
`collect('string')`	`['string']`	`false`
`collect(123)`	`[123]`	`false`
`collect([1, 2, 3])`	`[1, 2, 3]`	`false`
`collect(false)`	**`[false]`**	**`false`**

Performance

- benchmark for `filter`, `map` and `filter + map`
- steps 10, 100, 1000, 10000, 100000, 1000000 array entries
- 10 iterations with average runtime

Performance



karriere.at

Examples - Display Branches

- our company search uses branches for facetted search
- user wants to see selected branches on top of the search results

Examples - Display Branches

- our company search uses branches for facetted search
- user wants to see selected branches on top of the search results

```
$branches = [
  [
    'id' => 1,
    'name' => 'Branch A',
  ],
  [
    'id' => 2,
    'name' => 'Branch B',
  ],
  // ...
];

$filteredBranches = [1, 2];
```

karriere.at

Examples - Display Branches

```
$concatinatedBranches = collect($branches)
  ->whereIn('id', $filteredBranches)
```

Examples - Display Branches

```
$concatinatedBranches = collect($branches)
  ->whereIn('id', $filteredBranches)
  ->implode('name', ', ');

// $concatinatedBranches = "Branch A, Branch B"
```

Example - Notification types

- getting notification types for a user without a second database query

```
$notifications = [
  [
    'id' => 1,
    'type' => 'jobalarm',
    'message' => 'yada yada ...',
  ],
  [
    'id' => 2,
    'type' => 'company',
    'message' => 'karriere.at ...',
  ],
];
```

Example - Notification types

```
$notificationTypes = collect($notifications)
  ->pluck('type')
```

- `pluck` retrieves all values for the given key

Example - Notification types

```php
$notificationTypes = collect($notifications)
  ->pluck('type')
  ->unique();

// $notificationTypes = ['jobalarm', 'company'];
```

Example - get youngest notification

- get the timestamp of the newest jobalarm notification

```php
$notifications = [
  [
    'id' => 1,
    'type' => 'jobalarm',
    'message' => 'yada yada ...',
    'timestamp' => 1490080628,
  ],
  [
    'id' => 2,
    'type' => 'company',
    'message' => 'karriere.at ...',
    'timestamp' => 1490080523,
  ],
];
```

Example - get youngest notification

```
$timestamp = collect($notifications)
  ->where('type', '===', 'jobalarm')
```

Example - get youngest notification

```
$timestamp = collect($notifications)
  ->where('type', '===', 'jobalarm')
  ->pluck('timestamp')
```

Example - get youngest notification

```
$timestamp = collect($notifications)
  ->where('type', '===', 'jobalarm')
  ->pluck('timestamp')
  ->sortByDesc()
```

Example - get youngest notification

```
$timestamp = collect($notifications)
  ->where('type', '===', 'jobalarm')
  ->pluck('timestamp')
  ->sortByDesc()
  ->first();

// $timestamp = 1490080628;
```

- `$timestamp` will either contain the newest timestamp or `null` if no jobalarm notification is present.

Using Collections

- **Laravel** - built in since 5.0
- `illuminate/support` for multiple helpful features [1]
    - Collections
    - helper functions (array, strings, ...)
- `tightenco/collect` standalone package containing only collection features from `illuminate/support`

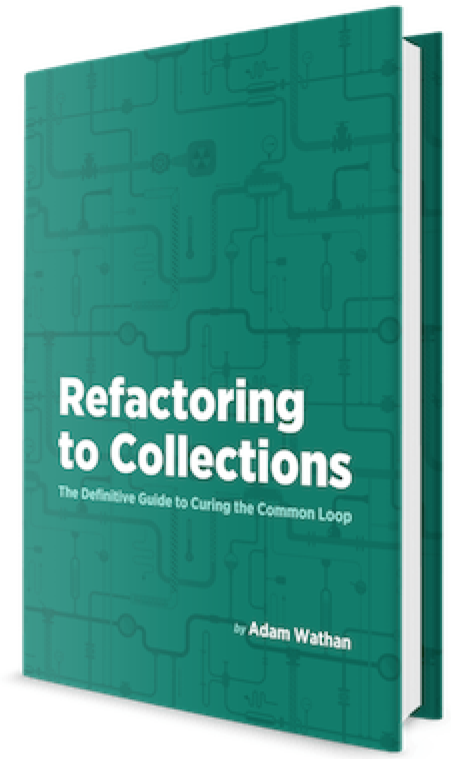
[1] https://github.com/illuminate/support

Wrap-up

- easily transform arrays to collections with `collect()`
- write pipeline style array manipulations
- null/empty handling out of the box
- descriptive way for array operations
- (almost) never write a loop again

Book recommendation

Refactoring to Collections
written by Adam Wathan
https://adamwathan.me/refactoring-to-collections/


Refactoring to Collections
The Definitive Guide to Curing the Common Loop
by Adam Wathan

karriere.at

Questions ?

karriere.at

We are hiring

- Data Scientist
- Software Test Engineer
- Full Stack Developer(s)
- and more (http://www.karriere.at/f/karriere-at/jobs)

karriere.at

Thank you

https://johannespichler.com

@fetzi_io

karriere.at