Colada — работа с коллекциями на новом уровне

Алексей Шоков

Пару слов обо мне

- Разработчик
  - более 6 лет опыта,
  - Symfony2/Doctrine2/...,
  - ZCE (Zend Certified Engineer)
- DevOps-евангелист
- Приверженец DDD
- ...

Зачем?

Заинтересовать функциональными подходом к работе с коллекциями, предоставив удобный инструмент и показав ход мыслей, приведших к его разработке.

Коллекции, они повсюду

- Конфигурация приложения
- Выборки из БД
- Пользовательские данные в HTTP-запросе
- ...

Функциональный подход

Определяем, ЧТО мы хотим сделать, а не КАК мы будем это делать.

Что мы имеем на текущий момент? В языке

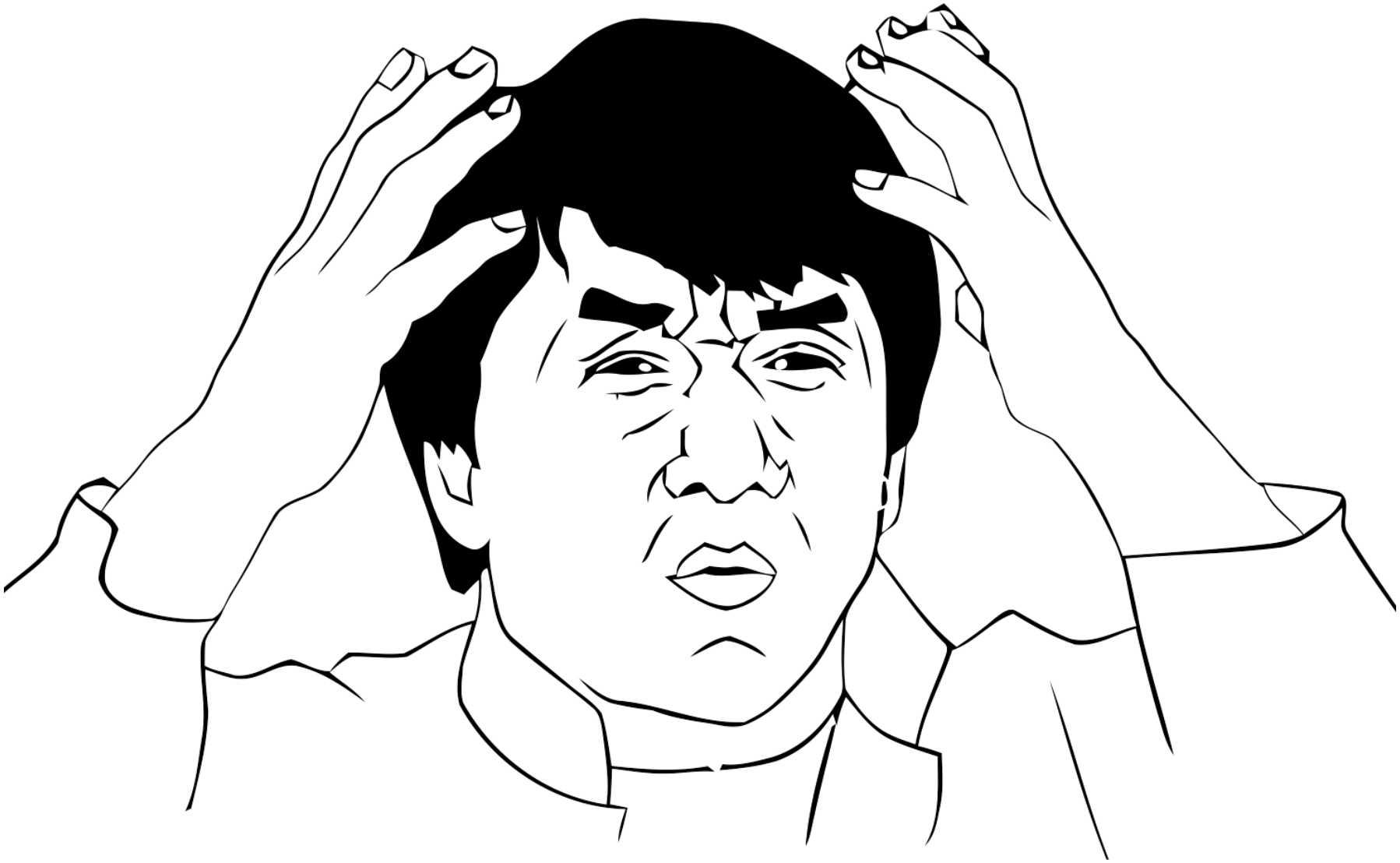
- `for ($i = 0; $i < count($users); $i++) { ... }` — legacy-код сразу после написания
- foreach ($users as $user) { ... } — уже лучше, но есть большой соблазн запихнуть все филтрации/преобразования в тело одного цикла, что значительно затруднит понимание при чтении в будущем

Что мы имеем на текущий момент? В стандартной библиотеке

- array **array_map** (callable *$callback* , array *$arr1* [, array *$...*])
- bool **array_walk** (array *&$array* , callable *$funcname* [, mixed *$userdata* = *NULL*])
- array **array_filter** (array *$input* [, callable *$callback* = *""*])
- bool **sort** (array *&$array* [, int *$sort_flags* = *SORT_REGULAR*])
- ...

Стандартная библиотека

- Разнобой в именовании функций и порядке принимаемых ими параметров
- Функциональные классы из SPL, которые слабо интегрируются между собой и никак не интегрируются с функциями для работы с массивами



Что мы имеем на текущий момент? Сторонние библиотеки

- Doctrine Collection — нет поддержки неизменяемых стрaктур данных, набор методов для обработки не велик
- PHP Collection (github.com/schmittjoh/php-collection) — те же самые недостатки
- Functional PHP (github.com/lstrojny/functional-php) — нет поддержки ООП (и нет поддержки неизменяемых стрaктур данных), богатый набор методов, имеется C-расширение
- ...и ещё несколько неподдерживающихся разработок

Чего хочется?

- Удобного интерфейса для работы
- Надёжного кода на выходе

Удобный интерфейс

- Богатый набор методов для обработки в едином стиле, поддерживающих вызов в цепочке
- Хеши с поддержкой различных типов ключей (не только скаляров)

Вдохновение

```
$ scala

scala> Seq(1,2,3)
res0: Seq[Int] = List(1, 2, 3)

scala> res0.filter(_ != 2).map(_ + 1)
res1: Seq[Int] = List(2, 4)

scala> res1.map(_.toString()).reduce(_ + ", " + _)
res2: String = 2, 4
```

Основные операции при обработке коллекций

- Фильтрация (acceptBy()/rejectBy())
- Преобразование (mapBy(), flatMapBy())
- «Схлопывание» (foldBy(), groupBy())

Например

```
$monitoredShops = $supervisors
    ->flatMapBy(x()->getShops())
    ->toSet();

$notMonitoredShops = $monitoredShops
    ->complement($shops)
    ->groupBy(x()->getCity());
```

x()

Для коротких функций преобразования/фильтрации пригодилась бы упрощённая запись, наример

```
$users->acceptBy(x()->isActive())
```

пишется/читается быстрее, чем

```
$users->acceptBy(function($user) {
    return $user->isActive();
})
```

Обработка коллекций

- head()/tail()
- sortBy()
- zip()/unzip()
- partitionBy()
- pluck()
- union()/intersection()/complement()
- ...

Хеши с поддержкой различных типов ключей (из жизни)

Задача: вывести таблицу отработанных часов по промоутерам подразделения за неделю.

```php
$checkins = ...; // Выборка из БД.

$builder = new MultimapBuilder(
    new CollectionBuilder(
        0, '...\\Entity\\CheckinCollection'
    ),
    '...\\Entity\\Timesheet'
);

foreach ($checkins as $checkin) {
    $builder->add($checkin->getPromoter(), $checkin);
}
```

```
$timesheet
    ->mapElementsBy(x()->groupByDay())
    ->eachBy(function($promoter, $checkins) {
        echo '<td>'.$promoter->getFullName().'</td>';
        $checkins->eachBy(function($checkin) {
            echo '<td>'
                .$checkin->getWorkedHours()
                .'</td>';
        })
        echo '<td>'.$checkins->getTotal().'</td>';
    });
```

Надёжный код

- Не ломается при любом наборе входных данных.
- Позволяет легко (часто) вносить изменеия, не боясь всё сломать (меньше возможностей прострелить себе ногу).

Что делает код надёжным?

Некоторые практики, относящиеся к работе с коллекциями, помогающие писать более надёжный код:

- использование неизменяемых структур данных,
- явное указание опциональных значений в интерфейсе.

Немодифицируемые коллекции

Работа разбивается на два этапа:

- создание коллекции
- и, собтвенно, дальнейшая работа с ней.

Создание

```php
$roles = collection('ROLE_USER', 'ROLE_ADMIN');

$config = to_map([
    'host' => 'yandex.ru',
    'port' => 80,
    'user' => null,
    'password' => null
]);
```

```php
$roles = (new SetBuilder())
    ->add('ROLE_USER')->add('ROLE_ADMIN')
    ->build();


$config = [
    'host' => 'yandex.ru',
    'port' => 80,
    'user' => null,
    'password' => null
];

$mapBuilder = new MapBuilder();
foreach ($config as $key => $value) {
    $mapBuilder->put($key, $value);
}

$config = $mapBuilder->build();
```

Опциональные значения (из жизни)

Реальный код:

```
echo $promoter->getShop()->getAddress();
```

Который «кладёт» страницу при отсутствии связанного объекта магазина (“Call to a member function on a non-object”).

Опциональные значения (из жизни)

При использовании опциональных значений (Colada) контракт используемого метода прозрачен:

```
echo $promoter->getShop()
    ->mapBy(x()->getAddress())
    ->orElse('N/A');
```

Опциональные значения — ближе к делу (хеши)

До боли знакомый каждому разработчику код:

```
$userRepository->findByPhone($request['phone']);
```

И возникающая в определённой ситуации ошибка:

```
Notice: Undefined index: phone in /var/www/index.php on line 1
```

Кто-то снова забыл проверку!

Опциональные значения — ближе к делу

Такая ситуация не возможна при использовании интерфейса, явно определяющего возможность отсутствия данных (опциональность) и предоставляющего инструменты для удобной обработки различных ситуаций.

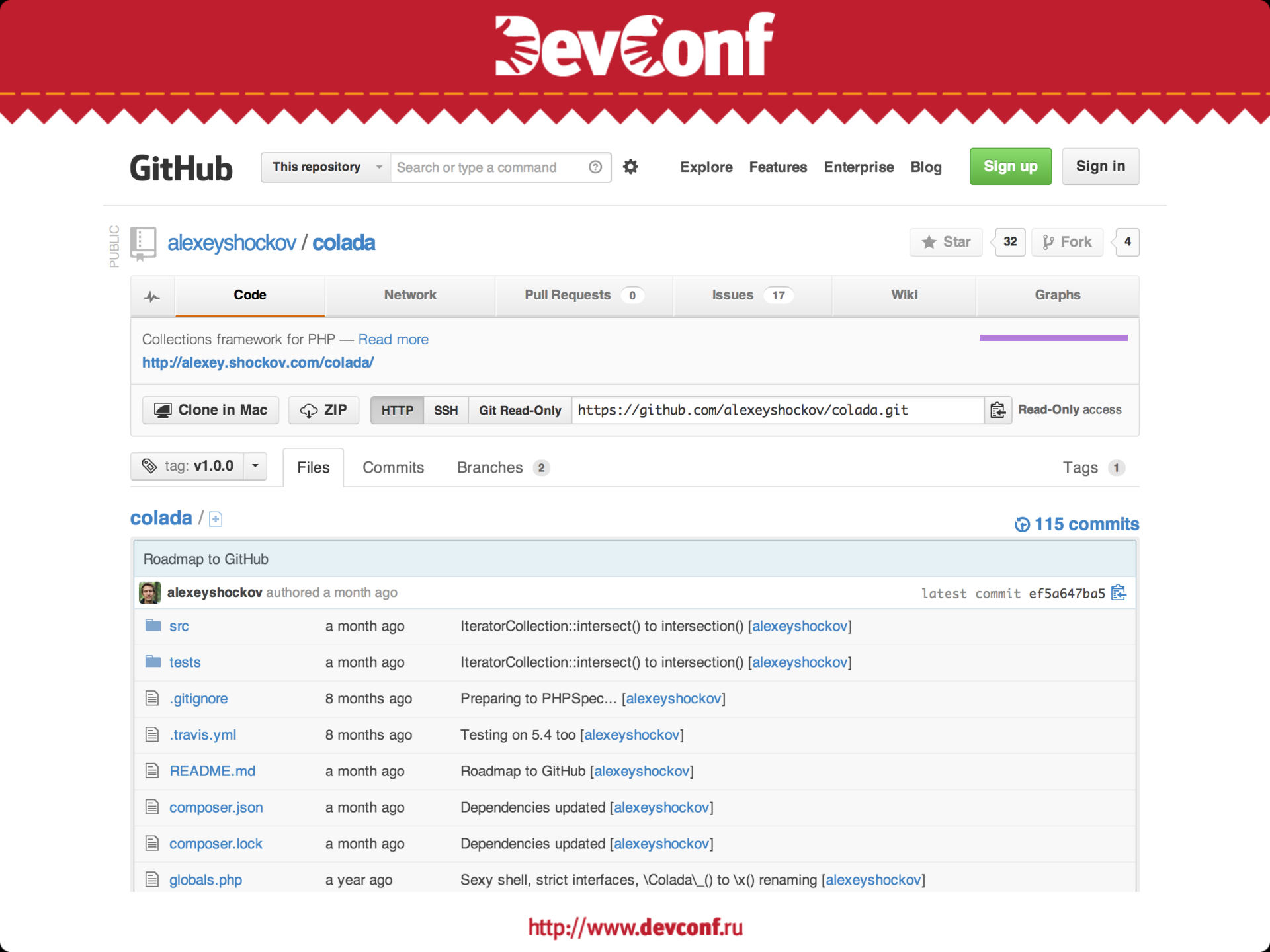
Опциональные значения — ближе к делу

```php
$userRepository->findByPhone($request['phone']->orThrow(
    new \IllegalArgumentException('Phone must be present.'))
);

$userRepository->findByActivity(
    $request['active']->mapBy('boolval')->orElse(true)
);
```

В результате

- Более читаемый код при использовании простых преобразований
- Меньше вероятность ошибок при использовании неизменяемых коллекций и опциональных значений
- Возможность оптимизации цепочки мелких операций (чтобы не потерять в производительности, или даже выиграть)


GitHub
This repository
Search or type a command
Explore
Features
Enterprise
Blog
Sign up
Sign in
PUBLIC
alexeyshockov / colada
Star 32
Fork 4
Code
Network
Pull Requests 0
Issues 17
Wiki
Graphs
Collections framework for PHP — Read more
http://alexey.shockov.com/colada/
Clone in Mac
ZIP
HTTP
SSH
Git Read-Only
https://github.com/alexeyshockov/colada.git
Read-Only access
tag: v1.0.0
Files
Commits
Branches 2
Tags 1
colada /
115 commits
Roadmap to GitHub
alexeyshockov authored a month ago
latest commit ef5a647ba5
src a month ago IteratorCollection::intersect() to intersection() [alexeyshockov]
tests a month ago IteratorCollection::intersect() to intersection() [alexeyshockov]
.gitignore 8 months ago Preparing to PHPSpec... [alexeyshockov]
.travis.yml 8 months ago Testing on 5.4 too [alexeyshockov]
README.md a month ago Roadmap to GitHub [alexeyshockov]
composer.json a month ago Dependencies updated [alexeyshockov]
composer.lock a month ago Dependencies updated [alexeyshockov]
globals.php a year ago Sexy shell, strict interfaces, \Colada_() to \x() renaming [alexeyshockov]

Colada

1.0

- Неизменяемые коллекции
- Опциональные значения
- Хеши с поддержкой ключей любого типа
- И всё, описанное выше :)

2.0 (скоро)

- Рефакторинг структуры классов (внутренние изменения)
- Пользовательская документация

Composer

```
{
    "require": {
        "alexeyshockov/colada": "1.*"
    }
}
```

Performance isn't issue (mostly)

В большинстве случаев разница в производительности некритична, т.к. код фильтрации/обработки элементов стоит намного больше накладных расходов по обходу коллекции.

В любом случае, в Colada мы имеем:

- экономию по памяти за счёт использования SplFixedArray (более чем в 2 раза по сравнению с array),
- «ленивое выполнение» основных методов фильтрации/обработки, что позволяет свести к минимуму проигрыш в скорости при большом количестве мелких операций.

Полёт сознания — DDD и коллекции

Логически, в приложении мы оперируем объектами модели предметной области, которые, в свою очередь, хранятся в коллекциях (одна большая коллекция по каждому типу объекта).

Tizona (github.com/alexeyshockov/tizona) — мысли на эту тему.

Спасибо за внимание!

github.com/alexeyshockov/colada — PR'ы приветствуются :)

@alexeyshockov