Watch your Clock

Andreas Heigl // 24.02.2022


12
1
2
3
9
10
11

- Team Lead at bitExpert (We're hiring 😉)
- Co-Organizer of PHPUGFFM
- OSS-Contributor
- Time-Nerd

Chris Seufert
an PHP Framework Interoperability Group

04.03.2021, 12:34:40

Hello,

I would like to propose standardizing a clock interface for reading the current time, date, timestamp, and timezone.

The problems I hope to solve is when testing time sensitive code, there is a unified way to represend the current time. I have run into several issues doing this some of which are:
- Some library use different Clock interfaces / implementations.
- Its impossible to mock some calls to date and time without using a PHP extension to override the current time & date.
- There are about 10 competing library implentations to this problem just from a quck search on packagist

The proposed interface would look like this:

```
interface ClockInterface
{
   public function timestamp():int;

   public function immutable():\DateTimeImmutable;

   public function datetime():\DateTime

    public function microtime():float

    public function timezone():\DateTimeZone
}
```

I have created a pull request for this here:
https://github.com/php-fig/fig-standards/pull/1224

I am looking to find out if anyone is interesting in sponsoring this Standard.

I have enjoyed using the PSR-16 interopable caches, and also migrated a large legacy application to use PSR-7/PSR-15 and also benefits from using standard middlewares and request extensions, and hope this could provide a similar experience in the future.

Regards,
Chris Seufert

Best Practices

The Problem

Filter all items that are in the future.

```php
/**
 * @var $iterable array<array-key, array{
 *   time: int,
 * }>
 */
$iterable = getItems();
```

Solution 1

```
$futureItems = array_filter($iterable, function ($item) {
    return time() < $item['time'];
});
```

Solution 2

```
function filterFutureItems(array $items); array
{
    return array_filter($items, function($item) {
        return time() < $item['time'];
    });
}

$futureItems = filterFutureItems($iterable);
```

```php
public function testFutureItemsAreFiltered(): void
{
    $items = [
        ['time' => time()-1],
        ['time' => time()],
        ['time' => time()+1],
    ];

    $newItems = filterFutureItems($items);
    $this->assertCount(1, $newItems);
}
```

Replace `time` and `date` etc with `DateTimeImmutable`

```php
// time();
(new DateTimeImmutable())->getTimestamp();

// echo date('c');
echo (new DateTimeImmutable())->format('c');
```

Updates Solution 2

```
function filterFutureItems(array $items); array
{
    return array_filter($items, function($item) {
        return (new DateTimeImmutable())->getTimestamp() < $item['time']
    });
}
```

improvement

```
function filterFutureItems(array $items); array
{
    $now = new DateTimeImmutable();
    return array_filter($items, function($item) use ($now) {
        return $now->getTimestamp() < $item['time'];
    });
}
```

Extract into class

```
final class FutureItemFilter
{
    public function filter(array $items): array
    {
        $now = new DateTimeImmutable();
        return array_filter($items, function($item) use ($now) {
            return $now->getTimestamp() < $item['time'];
        });
    }
}
```

Invoking the filter

```
$filter = new FutureItemFilter();
$items = $filter->filter($items);
```

Inject the filter

```
class Foo
{
    public function __construct(private FutureItemfilter $filter) {}

    public function doSomething()
    {
        $items = $this->getItems();
        $items = $this->filter->filter($items);
    }
}
```

further improvement

```
final class FutureItemFilter
{
    public function filter(array $items, DateTimeImmutable $now): array
    {
        return array_filter($items, function($item) use ($now){
            return $now->getTimestamp() < $item['time'];
        });
    }
}
```

Test

```php
public function testFutureItemsAreFiltered(): void
{
    $items = [
        ['time' => 1],
        ['time' => 2],
        ['time' => 3],
    ];

    $filter = new FutureItemsFilter();

    $newItems = $filter->filter($items, new DateTimeImmutable('@2'));
    $this->assertCount(1, $newItems);
}
```

Inject filter

```
class Foo
{
    public function __construct(private FutureItemfilter $filter) {}

    public function doSomething()
    {
        $items = $this->getItems();
        $items = $this->filter->filter($items, new DateTimeImmutable());
    }
}
```

Factories

Factory

```
final class CurrentDateTimeFactory
{
    public function getCurrentDateTimeObject(): DateTimeImmutable()
    {
        return new DateTimeImmutable();
    }
}
```

Improve filter further

```
final class FutureItemFilter
{
    public function __construct(private CurrentDateTimeFactory $factory)

    public function filter(array $items): array
    {
        $factory = $this->factory;
        return array_filter($items, function($item) use ($factory) {
            return $factory->getCurrentDateTime()->getTimestamp() < $ite
        });
    }
}
```

No change necessary in Foo

```php
class Foo
{
    public function __construct(private FutureItemfilter $filter) {}

    public function doSomething()
    {
        $items = $this->getItems();
        $items = $this->filter->filter($items);
    }
}

$f = new Foo(new FutureItemFilter(new CurrentDateTimeFactory()));
$f->doSomething();
```

Solution 3

Interface

```
interface DateTimeFactoryInterface
{
    public function getCurrentDateTimeObject(): DateTimeImmutable;
}
```

First implementation:

```
final class CurrentDateTimeFactory implements DateTimeFactoryInterface
{
    public function getCurrentDateTimeObject(): DateTimeImmutable()
    {
        return new DateTimeImmutable();
    }
}
```

Second Implementation:

```
final class TestDateTimeFactory implements DateTimeFactoryInterface
{
    public function __construct(private DateTimeImmutable $date) {}

    public function getCurrentDateTimeObject(); DateTimeImmutable)(
    {
        return $this->date;
    }
}
```

Adapted Filter

```
final class FutureItemFilter
{
        public function __construct(
            private DateTimeFactoryInterface $factory
        ) {}

    public function filter(array $items): array
    {
        $factory = $this->factory;
        return array_filter($items, function($item) use ($factory) {
            return $factory->getCurrentDateTime()->getTimestamp()
                   <
                   $item['time'];
        });
    }
}
```

```php
$filterForTestingPurposes = new FutureItemFilter(
    new TestDateTimeFactory(
        new DateTimeImmutable(
            '2022-02-22 12:23:34',
            new DateTimezone('America/Toronto')
        )
    )
);
$filterForRealLifeApplication = new FutureItemFilter(
    new CurrentDateTimeFactory()
);
```

Fin

Rename

```
-interface DateTimeFactoryInterface
+interface ClockInterface

-final class CurrentDateTimeFactory
+final class SystemClocck

-final class TestDateTimeFactory
+final class FixedClock
```

```
-public function getCurrentDateTime()
+public function now()
```

Why FIG

Be interoperable

History

PSR-0: Autoloading (2010)

PSR-4: Autoloading (2013)

PSR-3: Logging

PSR-6: Caching

How did I get involved

Initial Proposal

```php
interface CLockInterface
{
    public function timestamp(): int;
    public function immutable(): DateTimeImmutable;
    public function datetime(): DateTime;
    public function microtime(): float;
    public function timezone(): DateTimezone;
}
```

DateTimeImmutable is what DateTime should have been in the first place (Derick Rethans)

- public function timestamp(): int;
- public function immutable(): DateTimeImmutable;
- ~~public function datetime(): DateTime;~~
- public function microtime(): float;
- public function timezone(): DateTimezone;

- public function timestamp(): int;
- public function immutable(): DateTimeImmutable;
- ~~public function datetime(): DateTime;~~
- public function microtime(): float;
- ~~public function timezone(): DateTimezone;~~

- ~~public function timestamp(): int;~~
- public function immutable(): DateTimeImmutable;
- ~~public function datetime(): DateTime;~~
- public function microtime(): float;
- ~~public function timezone(): DateTimezone;~~

- ~~public function timestamp(): int;~~
- public function immutable(): DateTimeImmutable;
- ~~public function datetime(): DateTime;~~
- ~~public function microtime(): float;~~
- ~~public function timezone(): DateTimezone;~~

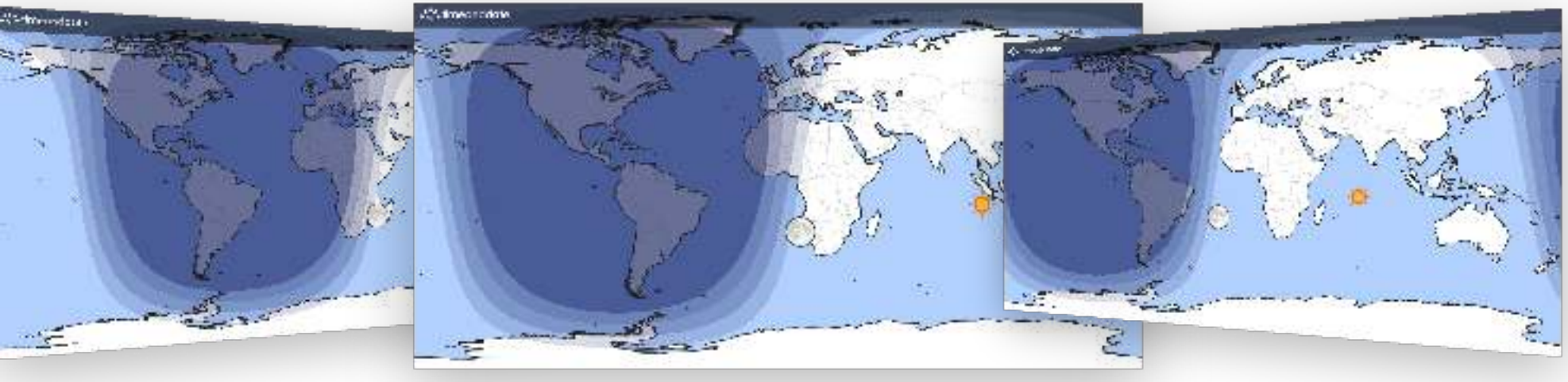
```php
interface ClockInterface
{
    public function now(): DateTimeImmutable;
}
```

Sleep

 Timezones!!

```php
$clock->now()->setTimezone(new DateTimezone('whatever you want'));
```

Excursion: What is "now"



05:30 UTC

```php
// When I created the slides
$a = new DateTimeImmutable(
    '2022-02-18T21:08:00',
    new DateTimeZone('Europe/Berlin')
);
// Now
$b = new DateTimeImmutable();

// Did I prepare the slides in Time?
var_dump($a < $b);
// true
```

Having to know the timezone of a datetime-object is a code-smell for me

```php
// Set clock to Montreal time
$clock = new SystemClock(new DateTimezone('America/Toronto'));
// Set clock to UTC
$clock = new SystemClock(new DateTimezone('UTC'));
```

Next Steps

- Bringing the PSR into Acceptance Vote
- Announce GA on Twitter/Discord/MailingList

```php
interface ClockInterface
{
    public function now(): DateTimeImmutable;
}

class Foo {
    public function __construct(private ClockInterface $clock) {}

    public function doSomething(): void
    {
        /* @var DateTimeImmutable */
        $now = $this->clock->now();
        // Do whatever needs to be done
    }
}
```

Questions?

Thank You!

Links:

- https://www.php-fig.org/psr/#draft
- https://github.com/php-fig/fig-standards/blob/master/propos
- https://github.com/php-fig/fig-standards/blob/master/propos
  meta.md
- https://packagist.org/providers/psr/clock-implementation
- https://groups.google.com/g/php-fig/c/lA8Lz0E4dWk/m/nvaf
- https://discord.gg/php-fig
- https://discord.com/channels/788815898948665366/81885
- https://twitter.com/phpfig