From annotations to attributes

Posted on November 4, 2022 by Grégoire Paris

Last month, we migrated the tests of the ORM from annotations to attributes. Let us look back on what lead to this moment.

Annotations

Let's go 22 years back in time. In October 2000, Ulf Wendel introduces phpdoc comments at the PHP-Kongress. These comments follow a structure that allows to produce API documentation from them. They are inspired by javadoc.

In 2002, Alex Buckley, a Specification lead for the Java language publishes JSR-175, thus proposing to add user-defined annotations to the language, allowing to tag language elements with extra information. 2 years later, it gets approved and Java 1.5, also known as Java 5 is released, with support for annotations.

4 more years elapse and in 2006, Jano Suchal publishes Addendum, a PHP library that adds support for using "Docblock/JavaDoc" as annotations, meaning that contrary to what is done in Java, Addendum annotations are contained inside phpdoc comments, like this:

```
/** @test */
function test_it_throws_on_invalid_argument(): void
{}

```

That is because they are implemented in userland, without requiring a change in PHP itself.

Doctrine ORM 2.0 is not released yet at that point, but the library is used to build an annotation driver in Doctrine 2 in early 2009. At that time, Doctrine was a project in a single repository, with `Common`, `DBAL` and `ORM` as top-level namespaces. Addendum is replaced 6 months later, with a new namespace under `Common` called `Annotations`.

In the summer of 2010, Guilherme Blanco and Pierrick Charron submit an RFC to add annotations support to PHP, but it gets declined. The RFC already mentions the need for annotations in PHPUnit, Symfony, Zend Framework, FLOW3 and of course, Doctrine.

Late 2010, Doctrine 2 is tagged, and the single repository is split into 3 repositories.

Finally, in 2013, the namespace above is isolated in its own repository, and `doctrine/annotations` 1.0.0 is tagged.

Today, the package is widely used in the PHP ecosystem and has a little short of 300 million downloads on Packagist, and is depended on by over 2 thousand packages, including major frameworks and tools. It is fair to say annotations have proven valuable to many users.

Attributes

The RFC mentioned above is only one among many. As mentioned before, annotations were implemented as phpdoc comments, which has several drawbacks:

- The comments are necessary to run the code, and need to be kept in the opcode cache.
- They are obtained at runtime, by using the reflection API, and because of that, can only be detected as invalid at runtime.
- They are not well supported by IDEs if at all.
- They clutter comments, which were originally intended for humans.
- They can be confused with phpdoc, which are something else entirely.

In March 2020, Benjamin Eberlei resurrects Dmitry Stogov's attributes RFC and submits the seventh RFC on this topic, introducing attributes to PHP.

A few rounds of RFCs about syntax later, PHP 8.0 is released, with a notable feature missing: nested attributes. PHP 8.0 attributes use a syntax that is forward-compatible with them though, and finally, with PHP 8.1, nested attributes are supported.

Top

Migrating from one to the other

Since attributes are much better than annotations, with `doctrine/orm` 3.0, annotations will no longer be supported, which means applications using them as a way to map entities to tables need to migrate towards attributes (or another driver). As maintainers of that library, even we needed to migrate: most of the test suite of `doctrine/orm` used annotations.

Enter Rector. Rector is a standalone tool that is invaluable when it comes to performing such migrations: it is able to understand PHP code and apply so-called Rectors to it. It is extensible, so it is possible to define such Rectors in order to address upgrades for anything, including Doctrine.

What's more, it comes with batteries included: when you install `rector/rector`, what you get is code from `rector/rector-src` ***and*** its official extensions, among which you will find `rector/rector-doctrine`. That's right, there is already an entire extension dedicated to Doctrine.

Rules are grouped together in sets, and the set that interests us here is `Rector\Doctrine\Set\DoctrineSetList::ANNOTATIONS_TO_ATTRIBUTES`.

To migrate `doctrine/orm`'s test suite to annotations, here is how we proceeded:

1. Install Rector: `composer require --dev rector/rector`.
2. Create a file called `rector.php` at the root of the library with the following contents:

```php
<?php

declare(strict_types=1);

use Rector\Config\RectorConfig;
use Rector\Doctrine\Set\DoctrineSetList;

return function (RectorConfig $rectorConfig): void {
    $rectorConfig->paths([
        __DIR__ . '/tests',
    ]);
    $rectorConfig->sets([
        DoctrineSetList::ANNOTATIONS_TO_ATTRIBUTES,
    ]);
};

```

3. Run `vendor/bin/rector`, which obeys the above configuration.
4. Uninstall Rector: `composer remove rector/rector && rm rector.php`
5. Run `vendor/bin/phpcbf` to make the migrated codebase compliant with our coding standard.

Or at least, it was the plan, because some annotations were not perfectly migrated. All in all, I found only 2 bugs, which looks great given how so many edge cases should appear in our test suite.

I went on and reported those 2 bugs, and this is where the experience went from great to stellar: the issue template leads to a playground, much like the one you can find for other tools such as Psalm or PHPStan.

This one comes with 2 buttons: "Create an issue", which pre-fills the Github issue with useful information, and "Create a test", that lets you create a test in the right directory (and also, the right repository, which is `rectorphp/rector-src`, and not `rectorphp/rector`).

If you want to add a new test for the bug you reported, you should let the official tutorial walk you through that, it is very well written.

Anyway, now that these 2 bugs are fixed and you know how to migrate, plan that migration, and let us know how it goes! Happy Rectoring!