Symfonyをアップグレードする方法

2016年12月14日 2020年8月26日

この記事は Symfony Advent Calendar 2016 の 14 日目の記事です。

Symfony3 がリリースされて 1 年経ちました。そろそろ 3 系へのアップグレードを考えている方も多いと思います。今回は、私がいつもやっているアップグレードの手順をまとめてみました。Symfony2.8 から 3.0 へのバージョンアップを例に紹介しますが、どのバージョンでも見るところは同じなので、いろんなバージョンに適用できる方法かと思います。

目次

Symfony Installer を使って 1 つ上のバージョンをダウンロードする

はじめに、Symfony Installer を使って 1 つ上のバージョンの Symfony をダウンロードします。例えば現在のバージョンが 2.8 の場合、3.0 をダウンロードします。この後たくさん diff を取るので、アップグレード対象の近くにダウンロードすると楽です。

Symfony Installer は Download Symfony Framework and Components を見て、あらかじめインストールしておきます。

```
$ symfony new symfony3.0 3.0
```

`symfony new` コマンドを実行すると、カレントディレクトリに symfony3.0 というディレクトリができます。

```
symfony3.0
├── README.md
├── app
├── bin
├── composer.json
├── composer.lock
├── phpunit.xml.dist
├── src
├── tests
├── var
├── vendor
└── web
```

composer.json を比較し、差分を取り込む

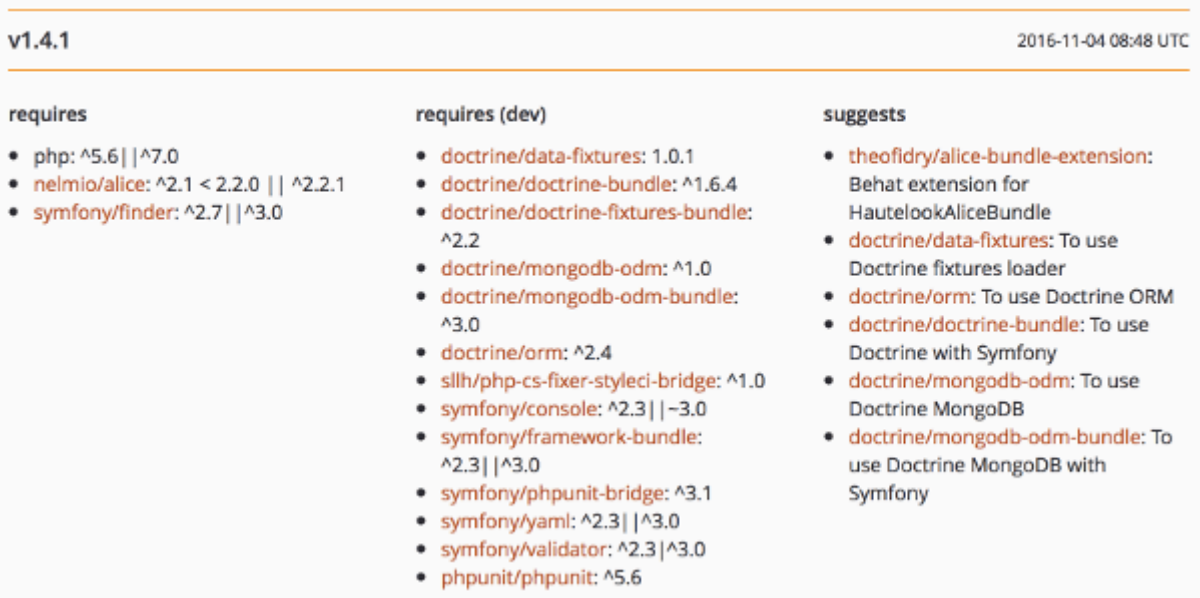
アップグレード対象の composer.json と、先程ダウンロードした symfony3.0 ディレクトリの composer.json を比較し、差分を取り込みます。

上記の画像は、とある Symfony2.8 のプロジェクトの composer.json と Symfony3.0 の composer.json を比較したところです。左が 2.8 右が 3.0 です。各バンドルのバージョンが変わっているのと、Symfony3 になって標準のディレクトリ構造が変わったため、autoload-dev などが差分として現れています。

サードパーティ製バンドルのバージョンを上げる

次に、composer.json に自身で追加したサードパーティ製のバンドルのバージョンを変更します。

Symfony 本体のバージョンが上がった場合 composer.json のバージョンをいくつにするべきか、いろいろ確認方法はあると思いますが、私は Packagist で確認することが多いです。



v1.4.1 2016-11-04 08:48 UTC

requires

- php: ^5.6||^7.0
- nelmio/alice: ^2.1 < 2.2.0 || ^2.2.1
- symfony/finder: ^2.7||^3.0

requires (dev)

- doctrine/data-fixtures: 1.0.1
- doctrine/doctrine-bundle: ^1.6.4
- doctrine/doctrine-fixtures-bundle: ^2.2
- doctrine/mongodb-odm: ^1.0
- doctrine/mongodb-odm-bundle: ^3.0
- doctrine/orm: ^2.4
- sllh/php-cs-fixer-styleci-bridge: ^1.0
- symfony/console: ^2.3||~3.0
- symfony/framework-bundle: ^2.3||^3.0
- symfony/phpunit-bridge: ^3.1
- symfony/yaml: ^2.3||^3.0
- symfony/validator: ^2.3|^3.0
- phpunit/phpunit: ^5.6

suggests

- theofidry/alice-bundle-extension: Behat extension for HautelookAliceBundle
- doctrine/data-fixtures: To use Doctrine fixtures loader
- doctrine/orm: To use Doctrine ORM
- doctrine/doctrine-bundle: To use Doctrine with Symfony
- doctrine/mongodb-odm: To use Doctrine MongoDB
- doctrine/mongodb-odm-bundle: To use Doctrine MongoDB with Symfony

上の画像は Packagist の hautelook/alice-bundle の v1.4.1 のページです。requires, requires(dev) のあたりを確認して Symfony3 系の対応を確認します。hautelook/alice-bundle は、nelmio/alice や、doctrine のバンドルに依存していますので、そのパッケージも Packagist で探して確認します。バンドルがバンドルに依存して、確認するバンドルが芋づる式に増えていくことがあるので、注意が必要です。

何かバンドルを導入するときはアップグレードのことも選定基準に入れておきましょう。

composer update コマンドを実行する

composer.json の変更が終わったら、 `composer update` コマンドを実行します。composer.json の書式や指定したバージョンに問題がなければライブラリが更新されます。post-update-cmd の clearCache のあたりでエラーが出て止まることが多いですが、この段階ではパッケージがすべて正常にダウンロードされていれば OK です。

標準のディレクトリ構造が変わっている場合は、新しい構造に合わせる

次にディレクトリ構造やファイルの変更に注目します。ディレクトリ構造が変わっている場合は、新しい構造に合わせます。

例えば Symfony3 では、標準のディレクトリ構造が変わり bin, var, tests といったディレクトリが追加されました。app/cache と app/logs は var 以下に移動になりました。app/phpunit.xml.dist は トップディレクトリ下に移動になりました。確認して変更を取り込みます。

標準ファイルを比較し、差分を取り込む

標準ファイルを比較し、差分をマージします。無くなったファイルがある場合は削除します。対象のファイルは大体こんな感じです。大変ですが 1 つずつ差分を取り込んでいきましょう。

```
symfony3.0
├── app
│   ├── AppCache.php
│   ├── AppKernel.php
│   ├── autoload.php
│   └── config
│       ├── config.yml
│       ├── config_dev.yml
│       ├── config_prod.yml
│       ├── config_test.yml
│       ├── parameters.yml
│       ├── parameters.yml.dist
│       ├── routing.yml
│       ├── routing_dev.yml
│       ├── security.yml
│       └── services.yml
├── bin
│   ├── console
│   └── symfony_requirements
├── phpunit.xml.dist
└── web
    ├── app.php
```

```
├── app_dev.php
└── config.php
```

Github symfony の UPGRADE-x.x.md を見ながら、既存コードに変更点を取り込む

GitHub の symfony/symfony にある UPGRADE-x.x.md を見ながら、変更点を取り込みます。

今回の例 2.8 から 3.0 へのアップグレードのとき、確認するファイルは symfony/UPGRADE-3.0.md です。

変更作業のイメージは、変更があったクラス名で src ディレクトリ以下を grep して、そのクラスを使っていたら変更点を適用していく感じです。

修正がすべて終わったら `composer run-script post-update-cmd` コマンドを実行してみます。先程実行した `composer update` コマンドのときと同様に、post-update-cmd の clearCacheエラーで止まることが多いです。メッセージを見ながら対処し、エラーが出なくなるまで繰り返します。

正常終了すると以下のように出力されます。

```
$ composer run-script post-update-cmd

ocramius/package-versions:  Generating version class...
ocramius/package-versions: ...done generating version class
> Incenteev¥ParameterHandler¥ScriptHandler::buildParameters
Updating the "app/config/parameters.yml" file
> Sensio¥Bundle¥DistributionBundle¥Composer¥ScriptHandler::buildBootstrap
> Sensio¥Bundle¥DistributionBundle¥Composer¥ScriptHandler::clearCache
 // Clearing the cache for the dev environment with debug true
 [OK] Cache for the "dev" environment (debug=true) was successfully cleared.
 ...
 ...
 [OK] All assets were successfully installed.
 ...
 ...
```

アプリケーションと PHPUnit を動かして動作確認

post-update-cmd が正常終了したら、実際にアプリケーションを動かしたり、PHPUnit を実行して動作確認をしましょう。エラーが出てきたら原因を見つけて対処します。

デバッグツールバーや PHPUnit の実行結果に deprecated 警告がでてくることがあります。UPGRADE-x.x.md の適用漏れかもしれませんので、見つけたら対処します。

問題なく動いていて、deprecated 警告に対処すれば、アップグレード完了です。

おまけ　マネージャーやステークホルダーの説得

本筋からちょっと離れた組織的な小話。でもエンジニアが自分たちのやりたいことをやるのに必要な話。

フレームワークのバージョンアップのような「売上に直接つながらないもの」について難色を示すマネージャー・ステークホルダーは多いと思います。システム全体に変更が入るのでリスクも高いですしね。マネージャー側の思考回路を知っておくと、このあたりの説得がしやすくなると思います。

マネージャーの頭の中（私の経験から。組織によってちょっと違うと思います）。

- 成果物を世の中に届けてお金を儲ける必要がある
- 納期、進捗、品質、報告、予算、リソース配分、中・長期的な計画、メンバーのキャリアプランとかモチベーションのことをずっと考えている
- 決断に責任を持つ立場なので、効果がよく分からないものにはリソースを投入できない（ステークホルダーに説明もできない）

これらを踏まえて、次のことを考えて持っていきます。

- バージョンアップしない場合、起こりうる問題
- バージョンアップの方法と超概算見積
- 作業に伴うリスク
- 費用対効果

リソース投資に対する効果を数値的に感じてもらうことが大事です。

アップグレードすることにより「長期的にみて開発の生産性を下げない」「セキュリティホールへの対応」「関わる人が幸せになる」ロジックを頑張って説明します。

まとめ

手順をざっくりおさらいすると、以下のようになります。

- composer.json の変更と composer update
- ディレクトリ構造の変更
- 標準ファイルの変更
- UPGRADE-x.x.md 対応

今回のように 2.8 から 3.0、3.0 から 3.1、3.2 へと、目的のバージョンまで 1 つずつ上げていくと問題の切り分けがしやすいです。一気にやらないのが大事。

アップグレードのコミットも他の人になぜやったか伝わるように、できるだけ細かく刻みましょう。

アップグレードするときは新機能の開発をせず、アップグレードに専念したほうが良いです。みんなで一気にやりましょう。