DOCTRINE
en la práctica

Antonio Garcia Elton Luís Minetto

Doctrine en la práctica

Elton Minetto y Antonio Garcia Marin

Este libro está a la venta en http://leanpub.com/doctrine-en-la-practica

Esta versión se publicó en 2016-06-06

Índice general

Introducción

Proyecto Doctrine

Doctrine es un proyecto *Open Source* que tiene por objetivo crear una serie de bibliotecas *PHP* para ofrecer funcionalidades de persistencia de datos y funciones relacionadas con ello.

El proyecto está dividido en varios sub-proyectos, siendo los dos más importantes el *Database Abstraction Layer* (*DBAL*) y el *Object Relational Mapper* (*ORM*).

Database Abstraction Layer

Construido sobre *PHP Data Objects* (*PDO*) *DBAL* proporciona una capa de abstracción que facilita la manipulación de datos usando una interfaz orientada a objetos. Para usar *PDO* es necesario tener las extensiones correspondientes configuradas. Si fuéramos a usar *DBAL* para acceder a una base de datos *MySQL*, por ejemplo, es necesario instalar la extensión correspondiente.

Además de la manipulación de datos (*insert*, *update*, etc) el paquete *DBAL* nos proporciona otras funcionalidades importantes como introspección de la base de datos (podemos obtener información sobre la estructura de tablas y campos), transacciones, eventos, etc. Vamos ver algunas de estas funcionalidades en próximos capítulos.

Object Relational Mapper

Voy usar aquí la definición que encontramos en la *Wikipedia* ya que resume bien el concepto de *ORM*:

> [...]es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia. Las tablas de la base de datos son representadas a través de clases y los registros de cada tabla son representados como instancias de las clases correspondientes. Con esta técnica, el programador no precisa preocuparse con los comandos en SQL; el usará una interfaz de programación simple que realiza todo el trabajo de persistencia.

Los *ORMs* son usados en muchos lenguajes de programación y entornos para facilitar el uso de base de datos y mantener una capa de abstracción entre diferentes bases de datos y conceptos.

Doctrine se ha convertido en un "standard de facto" para solucionar el problema de mapeamiennto objeto relacional en el entorno PHP y esta siendo utilizado por proyectos de diversos tamaños y frameworks como Symfony. A lo largo de los capítulos de este *e-book* aprenderemos a usarlo para este fin.

Instalación

La forma más fácil de instalar *Doctrine* es usando *Composer*. *Composer* es un gestor de dependencias para PHP. Con el puedes especificar que paquetes vamos usar en nuestro proyecto y gestionar la instalación y actualización de los mismos.

El primer paso es instalar el propio *Composer*. En *Linux* o *MacOSX* es posible instalar *Composer* por línea de comando, ejecutando el comando, dentro de directorio de nuestro proyecto:

```
curl -sS https://getcomposer.org/installer | php
```

Otra opción es ejecutar el siguiente comando, que no depende del paquete *curl*:

```
php -r "eval('?>'.file_get_contents('https://getcomposer.org/installer'));"
```

En *Windows* es posible descargar *Composer* de la url http://getcomposer.org/composer.phar[1] o usar el instalador binario, según la documentación oficial[2].

Con *Composer* instalado faltaría configurarlo para especificar que paquetes vamos usar. Para eso basta crear un archivo llamado *composer.json* en la raíz del proyecto. En nuestro caso vamos usar dos paquetes del proyecto *Doctrine* en sus versiones más recientes en el momento de la publicación de este libro, 2.4.X.:

```
{
    "require": {
        "doctrine/common": "2.4.*",
        "doctrine/dbal": "2.4.*",
        "doctrine/orm": "2.4.*"
    }
}
```

Puedes encontrar otros paquetes disponibles para *Composer* realizando una búsqueda en el directorio oficial de paquetes, que se encuentra en la web https://packagist.org[3].

Con el archivo *composer.json* creado podemos ejecutar el comando para que la instalación se realice:

[1] http://getcomposer.org/composer.phar
[2] http://getcomposer.org/doc/00-intro.md#installation-windows
[3] https://packagist.org

```
php composer.phar install
```

Creando el bootstrap.php

Ahora crearemos el *bootstrap* de nuestro proyecto. Este archivo posee este nombre puesto que es usado para inicializar y configurar el entorno. Se ejecutará todas las veces que ejecutamos algún script o página web, por eso es importante prestar especial atención a este archivo, para que no contenga errores o procesos pesados que puedan ralentizar la aplicación.

```php
<?php
//AutoLoader de Composer
$loader = require __DIR__.'/vendor/autoload.php';
//Añadimos nuestras clases al AutoLoader
$loader->add('DoctrineNaPratica', __DIR__.'/src');

use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;
use Doctrine\ORM\Mapping\Driver\AnnotationDriver;
use Doctrine\Common\Annotations\AnnotationReader;
use Doctrine\Common\Annotations\AnnotationRegistry;

//Si es FALSE se usa APC como cache, si fuese TRUE se arrays para la cache
$isDevMode = false;

//rutas de las entidades
$paths = array(__DIR__ . '/src/DoctrineNaPratica/Model');
// configuración de base de datos
$dbParams = array(
    'driver'   => 'pdo_mysql',
    'user'     => 'root',
    'password' => '',
    'dbname'   => 'dnp',
);

$config = Setup::createConfiguration($isDevMode);

//lector de las annotations de las entidades
$driver = new AnnotationDriver(new AnnotationReader(), $paths);
$config->setMetadataDriverImpl($driver);
//registra las annotations de Doctrine
AnnotationRegistry::registerFile(
```

```
33      __DIR__ . '/vendor/doctrine/orm/lib/Doctrine/ORM/Mapping/Driver/DoctrineAnno\
34  tations.php'
35  );
36  //Crea el entityManager
37  $entityManager = EntityManager::create($dbParams, $config);
```

https://gist.github.com/eminetto/7312206[4]

Intente documentar las principales funciones del archivo en los comentarios del código, pero voy a detallar algunos puntos importantes.

- Las primeras dos líneas de código son importantes pues cargan el *autoloader* de *Composer* y lo configuramos para reconocer las clases de proyecto, que crearemos a lo largo del libro. El *autoloader* es responsable de incluir los archivos PHP necesarios siempre que hagamos uso de las clases definidas en la seccion *use*.
- En la línea 18 definimos donde estarán las clases de nuestras *entidades*. En este contexto, entidades son la representación de las tablas de nuestra base de datos, que serán usadas por el *ORM* Doctrine. Creamos estas clases en el próximo capítulo.
- El código entre la línea 30 y la 36 es responsable de configurar las *Annotations* de Doctrine. Como veremos en el próximo capítulo existe más de una forma de configurar las entidades (*YAML* y *XML*) pero en este libro usaremos el formato de anotaciones de bloques de códigos, que es una de las formas más utilizadas.
- La línea 38 crea una instancia de *EntityManager*, que es el componente principal de *ORM* y como su nombre sugiere es el responsable de la manipulación de las entidades (creación, borrado, actualización, etc). Lo usaremos muchas veces a lo largo del libro.

Ahora necesitamos crear la estructura de directorios donde guardaremos las clases de nuestras entidades, conforme a lo configurado en la línea 18 de *bootstrap.php*. Esta estructura de directorios sigue el patrón *PSR*[5] que es usado por los principales frameworks y proyectos, inclusive el propio Doctrine. Con el siguiente comando, en Linux/MacOSX, creamos el directorio *src* dentro de la raíz de nuestro proyecto:

```
1  mkdir -p src/DoctrineNaPratica/Model
```

En la línea 20 de *bootstrap.php* configuramos Doctrine para conectar con una base de datos *MySQL* llamada *dnp*. Doctrine es capaz de crear las tablas representadas por las entidades, pero no puede crear la base de datos, ya que esto es algo que depende bastante de sistema gestor de base de datos. Por eso vamos crear la base de datos para nuestro proyecto, en *MySQL*:

[4] https://gist.github.com/eminetto/7312206
[5] https://github.com/php-fig/fig-standards/tree/master/accepted

```
mysql -uroot
create database dnp;
```

En el caso que este usando otro sistema gestor de base de datos como el *PostgreSQL*, *Oracle*, *SQLite*, etc, es necesario que verifique la necesidad o no de crear una base de datos antes de pasar al siguiente paso.

Configurar la herramienta de línea de comando

Uno de los recursos más útiles de Doctrine es su herramienta de línea de comando, que proporciona funcionalidades de gestión como crear tablas, limpiar cache, etc. El primero paso será crear el archivo de configuración de la herramienta, *cli-config.php*, en la raíz de nuestro proyecto:

```
<?php
// cli-config.php
require_once 'bootstrap.php';

$helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
    'db' => new \Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper($entityMana\
ger->getConnection()),
    'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($entityMa\
nager)
));
return $helperSet;
```

https://gist.github.com/eminetto/7312213[6]

Como podemos ver, se hace uso de *bootstrap.php* y crea una instancia de la clase *HelperSet* que es usada por el propio Doctrine, en la herramienta de línea de comandos.

Podemos probar si hemos configurado todo correctamente ejecutando:

```
./vendor/bin/doctrine
```

Si usamos Windows ejecutamos:

```
php vendor/bin/doctrine.php [1]
```

Se todo esta correcto veras una lista de comandos disponibles y una pequeña ayuda explicando como usarlo. Usaremos algunos de ellos en próximos capítulos.

(El uso de comillas dobles “” es obligatorio)

[6] https://gist.github.com/eminetto/7312213