



# DOCTRINE

na prática

Elton Luís Minetto

# **Doctrine na prática**

eminetto

Esse livro está à venda em <http://leanpub.com/doctrine-na-pratica>

Essa versão foi publicada em 2015-07-21



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

©2013 - 2015 eminetto

# Conteúdo

<b>Introdução . . . . .</b>	<b>1</b>
Projeto Doctrine . . . . .	1
<b>Instalação . . . . .</b>	<b>2</b>
Criando o bootstrap.php . . . . .	3
Configurar a ferramenta de linha de comando . . . . .	5

# Introdução

## Projeto Doctrine

O Doctrine é um projeto *Open Source* que tem por objetivo criar uma série de bibliotecas *PHP* para tratar das funcionalidades de persistência de dados e funções relacionadas a isto.

O projeto é dividido em alguns sub-projetos, sendo os dois mais importantes o *Database Abstraction Layer (DBAL)* e o *Object Relational Mapper (ORM)*.

### Database Abstraction Layer

Construído sob o *PHP Data Objects (PDO)* o *DBAL* provê uma camada de abstração que facilita a manipulação dos dados usando uma interface orientada a objetos. Por usar o *PDO* como suporte é necessário termos as extensões configuradas. Se formos usar o *DBAL* para acessar uma base de dados *MySQL*, por exemplo, é necessário instalar a extensão correspondente.

Além da manipulação de dados (*insert*, *update*, etc) o pacote *DBAL* nos fornece outras funcionalidades importantes como introspecção da base de dados (podemos obter informações da estrutura de tabelas e campos), transações, eventos, etc. Vamos ver algumas destas funcionalidades nos próximos capítulos.

### Object Relational Mapper

Vou usar aqui a definição que encontra-se na *Wikipedia* pois ela resume bem o conceito do *ORM*:

[...]é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes. Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; ele irá usar uma interface de programação simples que faz todo o trabalho de persistência.

Os *ORMs* são usados em diversas linguagens de programação e ambientes para facilitar o uso de banco de dados e manter uma camada de abstração entre diferentes bases de dados e conceitos.

O *Doctrine* tornou-se o “padrão de facto” para solucionar o problema do mapeamento objeto relacional no ambiente *PHP* e vem sendo usado por projetos de diversos tamanhos e frameworks como o *Symfony*. No decorrer dos capítulos deste *e-book* vamos aprender a usá-lo para este fim.

# Instalação

A forma mais fácil de instalarmos o *Doctrine* é usando o *Composer*. O *Composer* é um gerenciador de dependências para PHP. Com ele podemos definir quais pacotes vamos usar no nosso projeto e gerenciar a instalação e atualização dos mesmos.

O primeiro passo é instalarmos o próprio *Composer*. No *Linux* ou *MacOSX* é possível instalar o *Composer* pela linha de comando, executando o comando, dentro do diretório do nosso projeto:

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

Outra opção é executar o comando abaixo, que não depende do pacote *curl*:

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

No *Windows* é possível fazer o download do *Composer* pela url <http://getcomposer.org/composer.phar><sup>1</sup> ou usar o instalador binário, conforme mostra a [documentação oficial](#)<sup>2</sup>.

Com o *Composer* instalado precisamos configurá-lo para detalhar quais pacotes vamos usar. Para isso basta criarmos um arquivo chamado *composer.json* na raiz do projeto. No nosso caso vamos definir o uso dos pacotes do projeto *Doctrine* e suas versões mais atuais no momento da escrita deste livro, a 2.4.X.:

```
1 {
2     "require": {
3         "doctrine/common": "2.4.*",
4         "doctrine/dbal": "2.4.*",
5         "doctrine/orm": "2.4.*"
6     }
7 }
```

Você pode encontrar outros pacotes disponíveis para o *Composer* fazendo uma pesquisa no diretório oficial de pacotes, no site <https://packagist.org><sup>3</sup>.

Com o arquivo *composer.json* criado podemos executar o comando para que a instalação seja feita:

---

<sup>1</sup><http://getcomposer.org/composer.phar>

<sup>2</sup><http://getcomposer.org/doc/00-intro.md#installation-windows>

<sup>3</sup><https://packagist.org>

```
1 php composer.phar install
```

## Criando o bootstrap.php

Vamos agora criar o *bootstrap* do nosso projeto. Este arquivo possui este nome pois é usado para inicializar e configurar o ambiente, dar o “pontapé inicial” do projeto. Ele vai ser executado todas as vezes que executarmos algum script, por isso é importante especial atenção a este arquivo, para que ele não contenha erros ou processamentos pesados que possam deixar as coisas lentas.

```
1 <?php
2 //AutoLoader do Composer
3 $loader = require __DIR__ . '/vendor/autoload.php';
4 //vamos adicionar nossas classes ao AutoLoader
5 $loader->add('DoctrineNaPratica', __DIR__ . '/src');
6
7
8 use Doctrine\ORM\Tools\Setup;
9 use Doctrine\ORM\EntityManager;
10 use Doctrine\ORM\Mapping\Driver\AnnotationDriver;
11 use Doctrine\Common\Annotations\AnnotationReader;
12 use Doctrine\Common\Annotations\AnnotationRegistry;
13
14 //se for falso usa o APC como cache, se for true usa cache em arrays
15 $isDevMode = false;
16
17 //caminho das entidades
18 $paths = array(__DIR__ . '/src/DoctrineNaPratica/Model');
19 // configurações do banco de dados
20 $dbParams = array(
21     'driver'    => 'pdo_mysql',
22     'user'      => 'root',
23     'password'  => '',
24     'dbname'    => 'dnp',
25 );
26
27 $config = Setup::createConfiguration($isDevMode);
28
29 //leitor das annotations das entidades
30 $driver = new AnnotationDriver(new AnnotationReader(), $paths);
31 $config->setMetadataDriverImpl($driver);
32 //registra as annotations do Doctrine
```

```

33 AnnotationRegistry::registerFile(
34     __DIR__ . '/vendor/doctrine/orm/lib/Doctrine/ORM/Mapping/Driver/DoctrineAnno\
35 tations.php'
36 );
37 //cria o entityManager
38 $entityManager = EntityManager::create($dbParams, $config);

```

[https://gist.github.com/eminetto/7312206<sup>4</sup>](https://gist.github.com/eminetto/7312206)

Tentei documentar as principais funções do arquivo nos comentários do código, mas vou destacar alguns pontos importantes.

- As primeiras duas linhas de código são importantes pois carregam o *autoloader* do *Composer* e o configura para reconhecer as classes do projeto, que iremos criar no decorrer do livro. O *autoloader* é responsável por incluir os arquivos PHP necessários sempre que fizermos uso das classes definidas na sessão *use* dos arquivos.
- Na linha 18 definimos onde vão ficar as classes das nossas *entidades*. Neste contexto, entidades são a representação das tabelas da nossa base de dados, que serão usadas pelo *ORM* do Doctrine. Iremos criar estas classes no próximo capítulo.
- O código entre a linha 30 e a 36 é responsável por configurar as *Annotations* do Doctrine. Como veremos no próximo capítulo existe mais de uma forma de configurarmos as entidades (*YAML* e *XML*) mas vamos usar neste livro o formato de anotações de blocos de códigos, que é uma das formas mais usadas.
- A linha 38 cria uma instância do *EntityManager*, que é o componente principal do *ORM* e como seu nome sugere é o responsável pela manipulação das entidades (criação, remoção, atualização, etc). Iremos usá-lo inúmeras vezes no decorrer deste livro.

Precisamos agora criar a estrutura de diretórios onde iremos salvar as classes das nossas entidades, conforme configurado na linha 18 do *bootstrap.php*. Esta estrutura de diretórios segue o padrão *PSR*<sup>5</sup> que é usado pelos principais frameworks e projetos, inclusive o próprio Doctrine. No comando abaixo, do Linux/MacOSX, criamos o diretório *src* dentro da raiz do nosso projeto:

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

Na linha 20 do *bootstrap.php* configuramos o Doctrine para conectar em uma base de dados *MySQL* chamada *dnp*. O Doctrine consegue criar as tabelas representadas pelas entidades, mas não consegue criar a base de dados, pois isso é algo que depende bastante do sistema gerenciador de base de dados. Por isso vamos criar a base de dados para nosso projeto, no *MySQL*:

---

<sup>4</sup><https://gist.github.com/eminetto/7312206>

<sup>5</sup><https://github.com/php-fig/fig-standards/tree/master/accepted>

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

Caso esteja usando outro sistema gerenciador de banco de dados como o *PostgreSQL*, *Oracle*, *SQLite*, etc, é necessário que você verifique o a necessidade ou não de criar uma base de dados antes de passar para os próximos passos.

## Configurar a ferramenta de linha de comando

Um dos recursos muito úteis do Doctrine é a sua ferramenta de linha de comando, que fornece funcionalidades de gerenciamento como criar tabelas, limpar cache, etc. O primeiro passo é criarmos o arquivo de configuração da ferramenta, o *cli-config.php*, na raiz do nosso projeto:

```
1 <?php
2 // cli-config.php
3 require_once 'bootstrap.php';
4
5 $helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
6     'db' => new \Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper($entityManag
7 er->getConnection()),
8     'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($entityMa
9 nager)
10));
11 return $helperSet;
```

[https://gist.github.com/eminetto/7312213<sup>6</sup>](https://gist.github.com/eminetto/7312213)

Como podemos ver, ele faz uso do *bootstrap.php* e cria uma instância da classe *HelperSet* que é usada pelo próprio Doctrine, na ferramenta de linha de comandos.

Podemos testar se configuramos tudo da maneira correta executando:

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

Ou, no caso do windows:

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

Se tudo estiver correto você verá uma lista de comandos disponíveis e uma pequena ajuda explicando como usá-los. Iremos usar alguns deles nos próximos capítulos.

---

<sup>6</sup><https://gist.github.com/eminetto/7312213>