



andrewnester 6 января 2014 в 00:00

Как связать Yii Framework и Doctrine 2 ORM?

PHP, Yii, Doctrine ORM

Из песочницы



Мне очень нравится Yii Framework. Он быстрый, удобный, гибкий. Мне нравится, как реализован в нём паттерн ActiveRecord. Но бывают случаи, когда бизнес-логика, а, если быть точным, доменная логика, очень сложная и постоянно растёт и модифицируется. В таких случаях удобнее пользоваться паттерном DataMapper.

В тоже время мне нравится Doctrine 2 ORM. Это пожалуй самая мощная ORM для PHP, имеющая широчайший функционал. Да, возможно, она «тяжеловата» и замедляет работу приложения. Но начиная разработку, прежде всего стоит думать об архитектуре приложения, так как *«преждевременная оптимизация корень всех бед»*

Таким образом, однажды мне пришла в голову мысль связать 2 этих интересных мне инструмента. Как это было сделано, описано ниже.

Установка необходимых библиотек

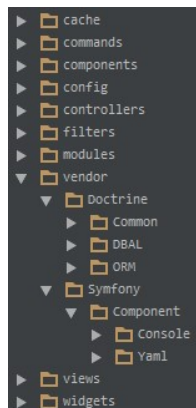
Связать Doctrine и Yii было решено с помощью создания соответствующего компонента DoctrineComponent, который бы и предоставлял доступ к функциям Doctrine.

Первым делом, в папке protected фреймворка была создана папка vendor, куда и был загружен код Doctrine 2 ORM. Установить Doctrine можно с помощью Composer либо просто скачав/склонировав исходники из [GitHub проекта Doctrine](#).

Также, для корректной работы ORM понадобятся [Doctrine Database Abstraction Layer](#) и [Doctrine Common](#) (при установке Doctrine 2 ORM с помощью Composer данные зависимости подтягиваются автоматически).

Кроме того, советую для того, чтобы была возможность работать с Doctrine 2 ORM через консоль установить в ту же папку vendor 2 компонента Symfony — это [Console](#) (для работы с Doctrine через консоль) и [Yaml](#) (при желании описания сущностей на Yaml)

Таким образом, на данном этапе должна быть получена следующая структура проекта:



Создание компонента DoctrineComponent

Теперь можно перейти непосредственно к созданию компонента DoctrineComponent. Ниже я приведу целиком код компонента, благо он достаточно небольшой. Данный код должен находиться в папке

Умные веса — твой помощник в контроле своего веса

[Как это?](#)

Реклама

ЧИТАЮТ СЕЙЧАС

Подсчитаем баги в калькуляторе Windows

11,6k 29

Красивые точные часы из старого смартфона

17,6k 53

Facebook отказывается переносить серверы в страны с нарушением свободы слова и вводит сквозное шифрование

709 7

Второй пациент с ВИЧ достиг ремиссии

8,5k 9

Анонсировали USB4: что известно о стандарте

3,5k 9

Жесткие диски в ноутбуках и настольных ПК могут служить источником прослушивания разговоров людей, находящихся рядом

10,2k 9

▼ Код файла DoctrineComponent.php

```

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

class DoctrineComponent extends CComponent
{
    private $em = null;
    private $basePath;
    private $proxyPath;
    private $entityPath;
    private $driver;
    private $user;
    private $password;
    private $host;
    private $dbname;

    public function init()
    {
        $this->initDoctrine();
    }

    public function initDoctrine()
    {
        Yii::setPathOfAlias('Doctrine', $this->getBasePath() . '/vendor/Doctrine');

        $cache = new Doctrine\Common\Cache\FilesystemCache($this->getBasePath() . '/cache');

        $config = new Configuration();
        $config->setMetadataCacheImpl($cache);

        $driverImpl = new AnnotationDriver(new AnnotationReader(), $this->getEntityPath());

        AnnotationRegistry::registerAutoloadNamespace('Doctrine\ORM\Mapping', $this->getBasePath() . '/vendor');

        $config->setMetadataDriverImpl($driverImpl);
        $config->setQueryCacheImpl($cache);
        $config->setProxyDir($this->getProxyPath());
        $config->setProxyNamespace('Proxies');
        $config->setAutoGenerateProxyClasses(true);
        $connectionOptions = array(
            'driver' => $this->getDriver(),
            'user' => $this->getUser(),
            'password' => $this->getPassword(),
            'host' => $this->getHost(),
            'dbname' => $this->getDbname()
        );

        $this->em = EntityManager::create($connectionOptions, $config);
    }

    public function setBasePath($basePath)
    {
        $this->basePath = $basePath;
    }

    public function getBasePath()
    {
        return $this->basePath;
    }

    public function setEntityPath($entityPath)
    {
        $this->entityPath = $entityPath;
    }
}

```

```

public function getEntityPath()
{
    return $this->entityPath;
}

public function setProxyPath($proxyPath)
{
    $this->proxyPath = $proxyPath;
}

public function getProxyPath()
{
    return $this->proxyPath;
}

public function setDbname($dbname)
{
    $this->dbname = $dbname;
}

public function getDbname()
{
    return $this->dbname;
}

public function setDriver($driver)
{
    $this->driver = $driver;
}

public function getDriver()
{
    return $this->driver;
}

public function setHost($host)
{
    $this->host = $host;
}

public function getHost()
{
    return $this->host;
}

public function setPassword($password)
{
    $this->password = $password;
}

public function getPassword()
{
    return $this->password;
}

public function setUser($user)
{
    $this->user = $user;
}

public function getUser()
{
    return $this->user;
}

/**
 * @return EntityManager
 */
public function getEntityManager()
{
    return $this->em;
}
}

```

Основная часть компонента заключена в методе `initDoctrine`. Разберём подробнее код.

```
$cache = new Doctrine\Common\Cache\FilesystemCache($this->getBasePath() . '/cache');
$config = new Configuration();
$config->setMetadataCacheImpl($cache);
```

Данным кодом мы устанавливаем метод кеширования метаданных сущностей из Doctrine. По-хорошему, тип кеширования (в данном случае `FilesystemCache`) следовало бы лучше вынести в параметры компонента, который мы могли бы менять при конфигурировании компонента.

```
$driverImpl = new AnnotationDriver(new AnnotationReader(), $this->getEntityPath());
AnnotationRegistry::registerAutoloadNamespace('Doctrine\ORM\Mapping', $this->getBasePath() .
    '/vendor');
$config->setMetadataDriverImpl($driverImpl);
```

С помощью кода выше устанавливается драйвер для чтения метаданных сущностей.

```
$config->setQueryCacheImpl($cache);
$config->setProxyDir($this->getProxyPath());
$config->setProxyNamespace('Proxies');
$config->setAutoGenerateProxyClasses(true);
```

Кодом выше мы устанавливаем метод кеширования для запросов (первая строка), остальные строки — настройка Proxy для Doctrine (путь, пространство имён, установка автоматического генерирования Proxy-классов)

```
$connectionOptions = array(
    'driver' => $this->getDriver(),
    'user' => $this->getUser(),
    'password' => $this->getPassword(),
    'host' => $this->getHost(),
    'dbname' => $this->getDbname()
);
$this->em = EntityManager::create($connectionOptions, $config);
```

Код выше определяет опции соединения с БД. Данные параметры задаются при подключении компонента (будет показано далее, как подключить компонент).

И в конце создаётся `EntityManager` с определёнными ранее `$connectionOptions` и `$config`, с помощью которого и можно работать с нашими сущностями.

Как подключить DoctrineComponent к проекту?

Перейдём к подключению `DoctrineComponent` к проекту.

Сделать этого довольно просто — необходимо просто внести изменения в конфигурационный файл проекта (обычно это `main.php`)

```
return array(
    'components' => array(
        'doctrine' => array(
            'class' => 'DoctrineComponent',
            'basePath' => __DIR__ . '/../',
            'proxyPath' => __DIR__ . '/../proxies',
            'entityPath' => array(
                __DIR__ . '/../entities'
            ),
            'driver' => 'pdo_mysql',
            'user' => 'dbuser',
            'password' => 'dbpassword',
            'host' => 'localhost',
            'dbname' => 'somedb'
        ),
    ),
    // ...
```

```
);
```

Теперь наш компонент будет доступен через `Yii::app()->doctrine`, а получить `EntityManager` мы можем через `Yii::app()->doctrine->getEntityManager()`

Но при таком использовании компонента возникает проблема в подсказках методов для объекта `EntityManager`. Для этого было придумано следующее решение:

```
class MainController extends Controller
{
    private $entityManager = null;

    /**
     * @return Doctrine\ORM\EntityManager
     */
    public function getEntityManager()
    {
        if(is_null($this->entityManager)){
            $this->entityManager = Yii::app()->doctrine->getEntityManager();
        }
        return $this->entityManager;
    }

    // ...
}
```

Каждый контроллер теперь наследуется от `MainController` и таким образом, в каждом контроллере можно вызвать метод `$this->getEntityManager()` для получения менеджера сущностей, причём в IDE теперь будут работать подсказки методов для `EntityManager`, что несомненно является плюсом.

Настройка консоли Doctrine

С Doctrine очень удобно работать через её консоль. Но для этого необходимо написать код для её запуска. Этот код приведён ниже. Я положил файл для запуска консоли в папку `protected/commands`. Очень хорошо также было бы реализовать команду `doctrine` для ещё более простого запуска консоли, но мной пока этого сделано не было.

Пример файл `doctrine.php` для работы с консолью Doctrine.

▼ [Код файла doctrine.php](#)

```
// change the following paths if necessary
$yii = __DIR__ . 'path/to/yii.php';
$config = __DIR__ . 'path/to/config/console.php';

require_once($yii);
Yii::createWebApplication($config);
Yii::setPathOfAlias('Symfony', Yii::getPathOfAlias('application.vendor.Symfony'));

$em = Yii::app()->doctrine->getEntityManager();
$helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
    'db' => new \Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper($em->getConnection())
),
    'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($em)
));

\Doctrine\ORM\Tools\Console\ConsoleRunner::run($helperSet);
```

Чтобы запустить консоль Doctrine достаточно перейти в папку `commands` и выполнить `php doctrine.php`.

Валидация моделей и использование моделей в виджете GridView.

Те, кто работал с Doctrine 2 ORM, знают, что фактически моделей в общепринятом их понятии (с методами валидации, получения данных из БД, включённой бизнес-логикой и т.д.) нет, а функциональность эта фактически разбита на 2 части — `Entity` и `Repository`. В `Entity` обычно включают бизнес-логику, а в `Repository` — методы получения данных из БД с использованием DBAL Doctrine (либо менеджер

сущностей, либо другим иным способом).

Валидация моделей

Таким образом, на мой взгляд, логично было бы включить валидацию данных в класс конкретной сущности. Рассмотрим на примере сущности `User`.

Чтобы не изобретать велосипед, было решено, что неплохо было бы использовать уже встроенную валидацию моделей из Yii, а конкретно из класса `CModel`.

Для этого просто-напросто можно наследовать сущность `User` от класса `CModel`. Пример такой сущности с описанными правилами валидации ниже:

▼ Код сущности User

```
use Doctrine\ORM\Mapping as ORM;

/**
 * User
 *
 * @ORM\Table(name="user")
 * @ORM\Entity(repositoryClass="UserRepository")
 * @ORM\HasLifecycleCallbacks
 */
class User extends CModel
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $id;

    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=255, nullable=false)
     */
    private $name;

    /**
     * @var string
     *
     * @ORM\Column(name="password", type="string", length=255, nullable=false)
     */
    private $password;

    /**
     * @var string
     *
     * @ORM\Column(name="email", type="string", length=255, nullable=false)
     */
    private $email;

    /**
     * @var string
     *
     * @ORM\Column(name="role", type="string", length=255, nullable=false)
     */
    private $role;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="created", type="datetime", nullable=false)
     */
    private $created;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="modified", type="datetime", nullable=false)
     */
    private $modified;
```

```

public function rules() {
    return array(
        array('name, password', 'required'),
        // ...
    );
}

public function attributeNames()
{
    return array(
        'id'=>'id',
        'name'=>'name',
        'email'=>'email',
        'created'=>'created',
        'updated'=>'updated'
    );
}

public function attributeLabels()
{
    return array(
        'description' => 'Description',
        'createdString' => 'Creation Date'
    );
}

// ...
}

```

Теперь приведу пример как с этой валидацией работать (пример создания нового пользователя ниже):

```

/**
 * Creates a new model.
 * If creation is successful, the browser will be redirected to the 'view' page.
 */
public function actionCreate()
{
    $user = new User();

    $userData = $this->getRequest()->get('User');
    $course->setAttributes($userData);
    if(!is_null($userData) && $user->validate())
    {
        $user->setName($userData['name']);
        // ... и так далее все поля

        $this->getEntityManager()->persist($user);
        $this->getEntityManager()->flush();
        $this->redirect(array('view', 'id'=>$user->getId()));
    }

    $this->render('create', array(
        'model'=>$user,
    ));
}

```

Использование моделей в виджете GridView

Одной из самых главных прелестей Yii являются, по-моему, виджеты, а в особенности различные Grid, которые идут в Yii из коробки.

Но единственный нюанс — они работают с ActiveRecord (я имею ввиду виджет GridView). А лично мне бы хотелось заставить их работать с Doctrine и сущностями. Для этого можно использовать Repository.

При использовании GridView есть 2 узких места — свойства `dataProvider` и `filter`. И здесь я пою оду разработчикам Yii — для того, чтобы GridView работал с какими-то данными, отличными от полученных из ActiveRecord, достаточно, чтобы объект, переданный в GridView в качестве `dataProvider` правильно реализовывал интерфейс `IDataProvider` (этот интерфейс и следует реализовать в нашем `UserRepository`), а объект, переданный в `filter`, — должен наследоваться от `CModel` (наша сущность `User` уже отлично

подходит для этого).

Всю реализацию `UserRepository` приводить не буду, обрисую только общую схему.

► [Код класса BaseRepository](#)

Выше пример реализации базового репозитория. Фактически реализацию многих методов можно посмотреть в Yii классе `CActiveDataProvider` который и реализует интерфейс `IDataProvider`. В `UserRepository` нам придётся определить лишь 2 метода(пример кода ниже):

```
<?php

class UserRepository extends BaseRepository
{
    protected $_id = 'UserRepository';

    /**
     * Fetches the data from the persistent data storage.
     * @return array list of data items
     */
    protected function fetchData()
    {
        //...
    }

    /**
     * Calculates the total number of data items.
     * @return integer the total number of data items.
     */
    protected function calculateTotalItemCount()
    {
        //...
    }
}
```

Резюме

Выше я привёл один из способов того, как можно работать в связке Yii + Doctrine 2 ORM. Многие могут сказать, что из-за Doctrine 2 ORM Yii потеряет свои преимущества, но не стоит забывать, что Doctrine имеет огромное количество средств для оптимизации и кеширования, да и никто не запрещает переписать слишком медленные либо интенсивные запросы на Plain SQL.

Зато в такой связке мы выигрываем в архитектурном решении и на мой взгляд, код становится от этого чище.

Был бы очень признателен, если бы в комментариях Вы поделились своими вариантами решения по внедрению паттерна DataMapper, каких-то других ORM в Yii, о своих способах решения разрастания бизнес логики в моделях ActiveRecord в Yii, о предметно-ориентированном программировании с использованием Yii.

Спасибо за внимание.

Теги: [Yii](#), [doctrine2](#), [php](#), [orm](#)

+13

95

17,7k

19



88,0

Карма

0,0

Рейтинг

272

Подписчики

Андрей Нестер [@andrewnester](#)

Software Engineer @ Amazon Web Services, AWSCloud9

[Github](#)

Поделиться публикацией



ПОХОЖИЕ ПУБЛИКАЦИИ

14 февраля 2016 в 23:26

Yii 2.0.7

+27 20,4k 44 7

18 августа 2015 в 11:15

Мой взгляд на файлы настроек yii 2

+10 16k 65 9

4 июня 2014 в 16:25

DevConf::PHP 2014 — уже на следующей неделе. PHPNG, Laravel, Yii, Асинхронный PHP — будь в курсе новинок разработки

+17 8k 30 17

ВАКАНСИИ

Мой круг



Senior Yii2 developer/Разработчик PHP Yii
ITGLOBAL.COM • Севастополь • Возможна удаленная работа

от 120000 до 200000 Р



PHP программист yii
Alma Innovation Group • Алматы • Возможна удаленная работа

от 50000 до 90000 Р



backend-разработчик (PHP/MySQL/yii)
Flowwow.com • Возможна удаленная работа

от 70000 Р



Backend-разработчик на PHP (Yii 2, удаленно)
Prosto.Insure • Возможна удаленная работа

от 50000 до 140000 Р



PHP Developer (PHP разработчик)
ISS • Москва

до 140000 Р

[Все вакансии](#)

Комментарии 19



mcavalon 6 января 2014 в 00:22

0

попробуем

OnYourLips 6 января 2014 в 12:23

0

Ни в коем случае. Мне такое даже в самых страшных кошмарах не приснится.

Многие могут сказать, что из-за Doctrine 2 ORM Yii потеряет свои преимущества, но не стоит забывать, что Doctrine имеет огромное количество средств для оптимизации и кеширования

Минусы доктрины не в производительности.



mcavalon 6 января 2014 в 12:24

0

Минусы доктрины не в производительности.

Просветите...

OnYourLips 6 января 2014 в 12:33

0

Много багов на ровном месте (к примеру, не всегда экранирует то, что должна, в результате чего получаются невалидные SQL-запросы), рыхлый код с повторениями, отсутствие целостности API, громоздкий код при использовании.

ЧТО ОБСУЖДАЮТ

Сейчас

Вчера

Неделя

Подсчитаем баги в калькуляторе Windows

11,6k 29

Про идеи преждевременные и своевременные на примере братьев Гракхов

2,6k 14

Актуальны ли пиратские FM радиостанции в наши дни?

25,5k 108

Facebook отказывается переносить серверы в страны с нарушением свободы слова и вводит сквозное шифрование

709 7

Fesor 6 января 2014 в 15:54

+1

Вы можете какую другую доктрину использовать, ибо никаких проблем ни с тем же экранированием, ни с кешированием, ни с невалидными SQL запросами я не встречал. Точно так же как не видел ненужного дублирования кода, отсутствия целостности API и громоздкого кода при использовании...

На поддержке остался только один проект на Yii к счастью, и мне больно использовать родной AR уже, не удобно... особенно миграции (привык я к migration:diff доктрины).

andrewnester 6 января 2014 в 12:32

0

а чем именно Вас не устраивает Doctrine 2?

и как я писал в заключении

Был бы очень признателен, если бы в комментариях Вы поделились своими вариантами решения по внедрению паттерна DataMapper, каких-то других ORM в Yii, о своих способах решения разрастания бизнес логики в моделях ActiveRecord в Yii, о предметно-ориентированном программировании с использованием Yii

как бы Вы решили такую проблему?

artemlight 6 января 2014 в 15:15

0

Когда-то давно я тоже столкнулся с подобной проблемой, и даже задавал этот вопрос на QA. toster.ru/q/46088#from_tracker Изначально в приложении было огромное количество плейнтекстовых sql-запросов, данные фетчились в ассоциативные массивы.

artemlight 6 января 2014 в 15:31

0

Извиняюсь, промахнулся мимо кнопки.

Когда-то давно я тоже столкнулся с подобной проблемой, и даже задавал этот вопрос на QA. toster.ru/q/46088#from_tracker Изначально в приложении было огромное количество плейнтекстовых sql-запросов, данные фетчились в ассоциативные массивы.

В итоге запилел свой собственный велосипед. Получилось весьма легковесно и быстро, работает автодополнение в IDE, кеширование, автовалидация, построчная загрузка из датасета и прочие полезняшки.

Идея не нова — для начала описываем объекты-входные параметры, объекты-выходные параметры. Резалтсет должен наследоваться от специального абстрактного класса.

```
class CCotoCustQueryParameters {
    function __construct()
    {
        $this->CUSTNO = new DBQuery\Parameters\CIntegerParameter();
    }
}

class CCotoCustQueryRecordSet extends DBQuery\ResultSet\ObjectResultSet {
    function __construct()
    {
        $this->CUST_CUSTNO = new DBQuery\Parameters\CIntegerParameter();
        $this->CUST_NAME = new DBQuery\Parameters\CStringParameter();
        $this->CUST_ADDR = new DBQuery\Parameters\CStringParameter();
        $this->CUST_WTEL = new DBQuery\Parameters\CStringParameter();
    }
}
```

... а затем используем эти объекты по прямому назначению

```
//Объявляем источник параметров
$coto_params = new SqlSchema\Service\CCotoCustQueryParameters();

//У нас параметр всего один - заполняем его из переданных в форму параметров
$coto_params->CUSTNO->setValue($reportParams->woCOTOCustNo->getValue());

//Источник SQL - плейнтекстовый файл, унаследован от CAbstractSource
$coto_sqlsource = new DBQuery\SqlSource\CTextFileSource('service/coto_cust');
```

```
//подгружаем в текстовый файл ранее забытые переменные
$coto_sqlsource->loadObject($coto_params);

//создаем экземпляр резултсета
$coto_resultset = new SqlSchema\Service\CCotoCustQueryRecordSet();

//Связываем источник sql и массив выходных резултсетов (их может быть больше одно
го, если столько отдаёт запрос)
$coto_query = new CDBQuery($coto_sqlsource,array($coto_resultset));

//Ну и выполняем
$coto_query->ExecutesSQL();
```

Теперь объект `$coto_resultset` можно просто передать в view. Конкретно этой вьюхи у меня сейчас нету, но есть другая. Там уже можно поиграться с форматированием, например. При наличии правильных тегов `phpdoc` автодополнение IDE работает и во вьюхах.

```
<td><?=$rs_workorders->GECNO->getValue() ?></td>
<td><?=$rs_workorders->CREATED->format(CDateTimeParameter::PhpOutputFormatDateOnly) ?></td>
<td><?=$rs_workorders->BILLD->format(CDateTimeParameter::PhpOutputFormatDateOnly) ?></td>
<td style='text-align: right'><?=$rs_workorders->SUM_WORK->formatMoney() ?></td>
<td style='text-align: right'><?=$rs_workorders->SUM_ITEM->formatMoney() ?></td>
```

Если кому интересно — могу рассказать об этом велосипеде подробнее, а то и в паблик выложить.



Fesor 6 января 2014 в 15:56

0

не вижу особо преимуществ даже перед родным AR, не то что перед доктриной.



Blumfontein 7 января 2014 в 01:46

0

Doctrine такое барахло. Что-то посложнее бложика и 90% запросов один фиг `QueryBuilder`. Элементарный запрос, где в `where` стоит что-либо отличное от «равно» — добро пожаловать `QueryBuilder`. Ладно бы еще Doctrine позволяла plain SQL прогонять, так нет же сиди настраивай `ResultSetMapping`. Кэширование — черный ящик какой-то, вроде все по tutorialу ставишь, а хрен пойми то ли кэширует, то ли не кэширует. Да и вообще не должна Модель заниматься кэшированием. `Translatable` — это вообще отдельный пи... ец, о котором даже говорить неохота. DQL — бредятина, вот не лень было придумывать еще абстракции над SQL.

Да и вообще в ORM реальная польза есть только от всяких связей `one-to-many`, `many-to-one`. Все остальное хлам. Работать с данными в виде объектов тот еще гемор: чуть-чуть нестандартный вывод в шаблоне — добро пожаловать велосипедить всякие функции для объектов, когда для массивов есть целая орда встроенных в язык функций.



Fesor 7 января 2014 в 02:04

+1

давайте по порядку:

Используйте DQL для выборки, это удобно. Используют же люди LINQ с SQL подобным синтаксисом. Выражения поддерживаются, можно объединять сложные выражения в свою функцию (например `GEODISTANCE`), в итоге у нас получаются красивая выборка из объектов для нашей бизнес логики, которой глубоко плевать откуда пришла эта выборка. В YII с этим дело обстоит если не так же то хуже. И ни в одной другой ORM.

По поводу plain SQL — так позволяет же. причем на выходе вы можете указать гидрацию не в объекты а в массив и разбирать все руками. Все эти `ResultMapping` они для более сложных кейсов.

По поводу кеширования — у вас модель ничего и не знает о нем. Этим занимается сервисный слой который сохраняет данные. Модель вообще ничего не знает где она хранится.

`Translatable` — не использовал в продакшене так что не могу ничего сказать. Но ничего страшного в нем не увидел. То что нужно для DDD и т.д.

ORM и созданы для работы со связями, так что ничего удивительного. Object-relation mapping как говорится из названия, это отображение данных в реляционных СУБД на ваши доменные-модели.

Главный плюс доктрины в том, что она вынесена целиком и полностью в сервисный слой. Это значит что вы можете без особых проблем в будущем заменить ORM скажем на ODM, не меняя при этом сами модели (или просто сменить ORM, или часть данных хранить где-то еще).

У вас есть только репозиторий ваших моделей, который по хорошему так же должен абстрагироваться от доктрины (имеется в виду что для каждой энити свой репозиторий реализующий свой интерфейс). Тогда достигается максимальный профит при дальнейшем расширении и поддержке.

Или вы хотите сказать что в Yii со всем что вы перечислили плохо? Или вы предлагаете полностью отказаться от ORM? С точки зрения бизнес логики у нас все равно останутся репозитории/провайдеры, а что они используют глубоко плевать.

 **Blumfontein** 7 января 2014 в 10:54

0

ДА я Yii-то и не защищаю, так как особо не работал с ним. А вот с доктриной опыт работы был, впечатления только отрицательные. Столько кода я даже на банальном QueryBuilder от какой-нибудь Kohana не писал. Вот меня все время возмущает, ну почему бы не сделать поддержку в методах find(), findAll() других знаков кроме "=". Вот надо сделать, скажем, условие с неравенством, сиди пиши queryBuilder.

Кстати, бесячий баг (или это фица?) поправили, когда для использования связи надо было спецом джойнить нужную модель в QueryBuilder. К примеру, есть модели Article и User, у Article есть связь на User (getUser). Если я напишу какой-нибудь запрос с QueryBuilder для Article и не заджойню явно User, то метод \$article->getUser() вернет NULL.

По поводу кеширования — у вас модель ничего и не знает о нем. Этим занимается сервисный слой который сохраняет данные. Модель вообще ничего не знает где она хранится.


А по-хорошему кеширование вообще надо вынести из доктрины, кешированием контроллер должен заниматься, а не доктрина.

Translatable — не использовал в продакшене так что не могу ничего сказать. Но ничего страшного в нем не увидел. То что нужно для DDD и т.д.

Открываете профайлер, смотрите на 100500 запросов, в ужасе закрываете.

Главный плюс доктрины в том, что она вынесена целиком и полностью в сервисный слой. Это значит что вы можете без особых проблем в будущем заменить ORM скажем на ODM, не меняя при этом сами модели (или просто сменить ORM, или часть данных хранить где-то еще).

Конечно БД мы каждый день меняем. Тяни ради этого громоздкую и тормозную библиотеку

 **Fesor** 7 января 2014 в 14:25

0

Я вообще стараюсь не использовать методы findAll или find или findBy. Как я уже говорил, репозитории у меня реализуют просто интерфейс, и зарегистрированы как сервисы (провайдера дынных по сути). Пока правда есть проблемы с менеджерами записей, контроллер все равно должен делать flush, с этим я пока не разобрался полностью.

Джойнить в любом случае нужно (вы же хотите сделать джойн, иначе какой в этом смысл). Но если вы в select не указали, то getUsers за счет ленивой подгрузки должен вернуть все. Хотя нужно проверить, но я не помню что бы у меня были какие-то проблемы с этим.

Кеширование в доктрине нужно для внутренних целей. Так что все в порядке. Именно кеширование запросов да — оно там не нужно, но я и не использую.

По поводу 100500 запросов — помню было дело, мне стандартный пагинатор (от KnpBundles) дико бесит, потому и отказался от него. Но в случае с Translatable вроде бы ничего особо страшного не приметил, увы не много с ним возился ибо потом было решено отказаться от хранения этого всего в MySQL в пользу монги (нужны были гео-индексы и индексация массивов по ключу).

По поводу смены базы данных — нет не каждый день. Но у меня были случаи когда часть данных уходило из локальной базы в базу клиента с доступом по API. В этом случае мне пришлось только переписать репозитории для нескольких сущностей и все.


Вообще на вкус и цвет. Мне нравится тот контроль который дает доктрина, мне нравится подход с мэппингом таблиц (объекты проецируются на базу а не наоборот). Если вам не нравится — не используйте. Но говорить о том что доктрина не нужна или сильно ущербна, это уж как-то слишком громко.

p.s. Так а что вы используете в конечном счете? Просто PDO/plain sql?

 **Blumfontein** 7 января 2014 в 16:48

0

Сейчас работаю с Rails, поэтому ActiveRecord. До этого кроме Symfony2 и Doctrine2 работал со встроенной в Kohana ORM, а также с обычным queryBuilder. Это на работе, для себя есть желание написать свой велосипед: взять обычный queryBuilder и немного расширить его функционал, всякие scope, relationship

 **zcasper** 29 марта 2018 в 05:33

0

> о своих способах решения разрастания бизнес логики в моделях ActiveRecord в Yii

А зачем её там вообще хранить?



Fesor 29 марта 2018 в 23:58

0

потому что это определение active record? Ну то есть если мы убираем всю бизнес логику из AR моделей, и оставляем там только тупое хранение данных и доступ к рядам таблиц через объектную прослойку — это будет уже не AR. У этого подхода есть свое имя — row data gateway, и это вполне себе норм практика.

естественно что в AR будет лежать только та бизнес логика, которая непосредственно относится к самой записи а не вообще вся бизнес логика. Это было бы глупо.



zcasper 2 апреля 2018 в 18:22

0

Как бы да, но я склонялся в ответе к тому, что MVC это всё же Presentation layer, а не Domain или Data source. Для простоты наверное в этом фрейме AR поселили в M...



Fesor 2 апреля 2018 в 22:10

0

Рекомендую: [Data-Context-Interaction](#)



zcasper 4 апреля 2018 в 17:18

0

)))) (like)

Только [полноправные пользователи](#) могут оставлять комментарии. [Войдите](#), пожалуйста.

САМОЕ ЧИТАЕМОЕ

Сутки

Неделя

Месяц

Microsoft открыла код Калькулятора Windows

+52

49,3k

62

143

Бобинники: десять культовых катушечных магнитофонов — рассказываем и показываем

+26

55,8k

34

24

Российская компания запустила серийное производство нейропроцессоров — конкурентов Nvidia

+32

28,4k

39

105

Красивые точные часы из старого смартфона

+72

17,6k

101

53

Наша с девушкой первая видеоигра. Разработка на Unity. Часть 1

+72

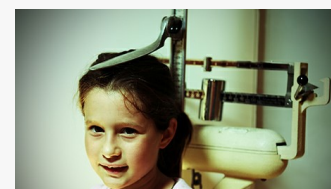
17,5k

90

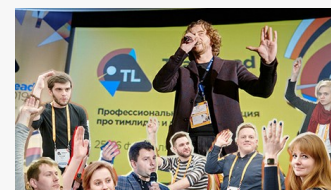
26

ХАБР РЕКОМЕНДУЕТ

[Разместить](#)



Подросли: во что превратились гаджеты нашего детства



Организовать и возглавить: полезные кейсы с TeamLead Conf 2019

Аккаунт

Разделы

Информация

Услуги

Приложения

[Войти](#)

[Регистрация](#)

[Публикации](#)

[Хабы](#)

[Компании](#)

[Пользователи](#)

[Песочница](#)

[Правила](#)

[Помощь](#)

[Документация](#)

[Соглашение](#)

[Конфиденциальность](#)

[Реклама](#)

[Тарифы](#)

[Контент](#)

[Семинары](#)

