PHP - Frameworks

Object-Relational-Mapping

Erik Brändli

15. Mai 2016

Contents

1 Einleitung

Sinn dieser Ausarbeitung ist es Orm - Frameworks (oder kurz ORMs) zu untersuchen und zu vergleichen hinsichtlich ihrer Funktionalität. Folgende Frameworks wurden ausgewählt für die Untersuchung: Doctrine und Propel.
Die Server wurden in Docker provisioniert.

2 Aufbereitung

ein lokales Verzeichnis wurde erstellt namens "*docker-orm*". In diesem habe ich dann einen php-container erstellt.

Listing 1: Installation der php-Umgebung

```
$ docker pull php:5.6-apache
$ docker run -d -p 80:80 --name php-orm-frameworks -v "$PWD":/var/www/html
    php:5.6-apache
$ docker exec -ti php-orm-frameworks /bin/bash
(php-orm #) cd
(php-orm #) mkdir opt
(php-orm #) apt-get install git
(php-orm #) apt-get install php5-pgsql
(php-orm #) apt-get install php5-sqlite
(php-orm #) apt-get install php5-mysql
apt-get install wget
```

3 Propelorm's Propel2

Propel2 ist ein OpenSource ORM-Framework welches aktiv entwicket wird. Meist ist der letzte Commit auf Github wenige Stunden her. Außerdem hat es eine Test-Coverage von 89.7%.

3.1 Installation

Listing 2: In der bash am PHP-Container

```
cd opt
git clone https://github.com/propelorm/Propel2 vendor/propel
wget http://getcomposer.org/composer.phar
php composer.phar install
export PATH=$PATH:/root/opt/vendor/propel/bin
propel -V
>> Propel version 2.0.0-dev
```

3.2 Projektaufsetzung

3.3 Definition eines Datenbankschematas

Listing 3: Definition eines Schemas ⟶ schema.xml

```
<database name="default" defaultIdMethod="native" defaultPhpNamingMethod="underscore">
  <table name="auftrag" idMethod="native" phpName="Auftrag">
    <!-- column name="firmenname" phpName="Firmenname" type="VARCHAR" size="50"
        primaryKey="true" required="true"/-->
    <column name="nummer" phpName="Nummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <column name="datum" phpName="Datum" type="DATE"/>
    <column name="status" phpName="Status" type="VARCHAR" size="50"/>
  </table>
  <table name="bedient" idMethod="native" phpName="Bedient">
    <column name="mitnummer" phpName="Mitnummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <column name="maschnummer" phpName="Maschnummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <foreign-key foreignTable="maschine" name="bedient_maschnummer_fkey">
      <reference local="maschnummer" foreign="nummer"/>
    </foreign-key>
    <foreign-key foreignTable="mitarbeiter" name="bedient_mitnummer_fkey">
      <reference local="mitnummer" foreign="nummer"/>
    </foreign-key>
  </table>

  <table name="maschine" idMethod="native" phpName="Maschine">
    <column name="nummer" phpName="Nummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <column name="beschreibung" phpName="Beschreibung" type="VARCHAR" size="30"/>
  </table>
  <table name="mitarbeiter" idMethod="native" phpName="Mitarbeiter">
    <column name="nummer" phpName="Nummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <column name="einstellungsdatum" phpName="Einstellungsdatum" type="DATE"/>
    <foreign-key foreignTable="person" name="mitarbeiter_nummer_fkey">
      <reference local="nummer" foreign="nummer"/>
    </foreign-key>
  </table>
  <table name="person" idMethod="native" phpName="Person">
    <column name="nummer" phpName="Nummer" type="INTEGER" primaryKey="true"
        required="true"/>
    <column name="vorname" phpName="Vorname" type="VARCHAR" size="30"/>
    <column name="nachname" phpName="Nachname" type="VARCHAR" size="30"/>
  </table>
</database>
```

3.3.1 Tabellendefinition

phpName	Name der repräsentativen Klasse der Tabelle
schema	DB-Schema der Tabelle
skipSql	ertellt kein DDL-Skript
abstract	Wie in SEW $\longrightarrow$ Vererbung
baseClass	falls eine andere Klasse als propel.om.BaseObject verwendet werden soll
heavyIndexing	Indexiert jeden PK (bei zusammengesetzten Pks)
identifierQuoting	Falls man reservierte Bezeichnungen benutzt
readOnly	Verhindert Schreibzugriffe
reloadOnInsert	Falls Insert Trigger verwendet werden (Daten reload)
reloadOnUpdate	Falls Update Trigger verwendet werden (Daten reload)

PK . . . Primary Key

3.3.2 Spaltendefinition

type	Datentyp
defaultValue	Standardwert dieser Spalte
valueSet	Vordefinierte Werte die zulässig sind
lazyLoad	Ob der Wert nicht mit-gefetcht werden soll (BLOB & CLOB)
autoIncrement	Autoincrement
primaryKey	Ob diese Spalte zum PK gehört
required	Equivalent zu "not null"
size	Z. B.: Textlänge

Folgende Datentypen werden aktzeptiert:

BOOLEAN	TINYINT	SMALLINT	INTEGER	BIGINT	DOUBLE
FLOAT	REAL	DECIMAL	CHAR	VARCHAR	LONGVARCHAR
DATE	TIME	TIMESTAMP	BLOB	CLOB	ARRAY

Foreign Keys werden in einem eigenen Tag spezifiziert "*foreign-key*"

3.3.3 Foreign Keys

Listing 4: Foreign Keys referenzieren

```
<foreign-key foreignTable="person" name="mitarbeiter_nummer_fkey">
     <reference local="nummer" foreign="nummer"/>
</foreign-key>
```

foreignTable	Entfernte Tabellenname
local	lokaler Spaltenname
foreigen	entfernter Spaltenname
name	Constraintbezeichnung
reference	Spalte des Constraints

OTO . . . One-To-One
OTM . . . One-To-Many
MTM . . . Many-To-Many

OTO wurde ermöglicht durch die Referenzierung von 2 PKs gegeneinander. Sprich wenn der

eine PK ein FK von einem PK ist.

OTM wird referenziert wenn man auf eine nicht PK-Spalte referenziert.

Bei MTM arbeitet man mit Kreuztabellen, diese haben beide Attribute referenziert als PK & FK.
außerdem muss man im ***table*** Tag *isCrossRef="true"* spezifizieren.

3.4 Verbindungskonfiguration

Listing 5: Verbindungsspezifikation

```
propel:
  database:
      connections:
          default:
              adapter: mysql
              classname: Propel\Runtime\Connection\DebugPDO
              dsn: mysql:host=127.0.0.1;dbname=fiddle_b027507
              user: b027507
              password: 4bc6a82eb7bbba
              attributes:
  runtime:
      log:
          defaultLogger:
              type: stream
              path: ./propel_log.txt
              level: 100
      defaultConnection: default
      connections:
          - default
  generator: #reverse-enginnering der Datenbank
      defaultConnection: default
      connections:
          - default
```

Mögliche Konfigurationsformate:

PHP	INI	YAML	XML	JSON

3.5 Anwendungscode Beispiele

Listing 6: Verbindungsspezifikation

```
<?php

$autoloader = require '/vendor/autoload.php';
$autoloader->add('', __DIR__ . '/generated-classes/');
require './generated-conf/config.php';

$a1 = new Auftrag();
$a1->setFirmenname("Whatever");
$a1->setNummer(1);
$a1->setStatus("aktzeptiert");
$a1->save();

$a1 = new Auftrag();
$a1->setFirmenname("Whatever2");
$a1->setNummer(2);
$a1->setStatus("abgelehnt");
$a1->save();

$auftraege = AuftragQuery::create()
  ->filterByNummer(array('min' => 1))
  ->find();
echo count($auftraege);
foreach($auftraege as $a1){
 $a1->setNummer(0);
 $a1->save();
}
$auftraege->save();


$ae2 = AuftragQuery::create()
  ->filterByStatus('abge%')->update(array('Nummer' => -1));
```

4 Doctrine

Doctrine besitzt eine DQL (vergleichbar mit Hibernates HQL). Es hat eine coverage von 86%.

4.1 Installation

Listing 7: composer.json für Doctrine

```
{
    "require": {
        "doctrine/orm": "*"
    },
    "autoload": {
        "psr-0": {"": "src/"}
    }
}
```

Mittels "*composer install*" oder "*php composer.phar install*" installieren. Um "composer.phar" zu erhalten "*wget https://getcomposer.org/composer.phar*".

4.2 Konfiguration

Listing 8: Konfiguration Doctrine ⟶ bootstrap.php

```
<?php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for Annotations
$isDevMode = true;
$config = Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"),
    $isDevMode);
// or if you prefer yaml or XML
//$config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"),
    $isDevMode);
//$config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"),
    $isDevMode);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_pgsql',
    'user' => 'root',
    'password' => 'toor',
    'host' => '172.17.0.2',
    'dbname' => 'doctrine',
    'charset' => 'UTF-8'
);

// obtaining the entity manager
$entityManager = EntityManager::create($conn, $config);
```

Man kann auch eine Verbindungs-URL verwenden: "*pgsql://172.17.0.2:5432/doctrine?charset=UTF-8*"
"*$config = Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"), $isDevMode);*"
legt fest, dass im Ordner './src' nach den Entitätsklassen gesucht wird.

Listing 9: Konfiguration Doctrine ⟶ cli-config.php

```
<?php
/**
 * Created by PhpStorm.
 * User: fusions
 * Date: 20.05.16
 * Time: 20:57
 */
require_once "bootstrap.php";

return \Doctrine\ORM\Tools\Console\ConsoleRunner::createHelperSet($entityManager);
```

4.2.1 Entitäten

Listing 10: Beispiel Entität Kunde ⟶ src/Kunde.php

```
<?php

/**
 * Kunden-Entity
 * @Entity @Table(name="t_customer")
 **/
class Kunde
{
   /** @Id @Column(type="integer") @GeneratedValue **/
    protected $Id;
    /** @Column(type="string") **/
    protected $firstname;
    /** @Column(type="string") **/
    protected $lastname;
    /** @Column(type="string") **/
    protected $address;

    /**
     * @return mixed
     */
    public function getId()
    {
        return $this->Id;
    }

    /**
     * @return mixed
     */
    public function getFirstname()
    {
```

```
        return $this->firstname;
    }

    /**
     * @param mixed $firstname
     */
    public function setFirstname($firstname)
    {
        $this->firstname = $firstname;
    }

   // snip
}
```

In doctrine können Entitäten per Annotation, XML oder YAML definiert werden.
Wichtig ist bei den Annotationen das man ” (Doppelhochkomma) verwendet werden.

4.2.2 Foreign Keys

Dabei ist wichtig dass man auf bi-direktionalität achtet!
Bei höheren als OTO (dh. OTM MTM) wird dies in den PHP Klassen durch Collections gelöst.

Listing 11: Doctrine Relationen ⟶ src/Kunde.php

```
   /**
     * @ManyToMany(targetEntity="Product")
     **/
    protected $products;


public function __construct()
    {
        $this->products = new ArrayCollection(); /** Zb products die sich der Kunde in
            den Waren korb legte **/
    }
```

4.3 Code-Beispiel

Listing 12: Doctrine Relationen ⟶ src/Kunde.php

```
$user = new User();
$user->setName("Max");
$user->setPassword("test");
$entityManager->persist($user);
$entityManager->flush();
echo "Der Benutzer mit der ID ".$user->getId()." wurde erfolgreich hinzugefuegt.";
```

5 Vergleich

Die meisten WebFrameworks ermöglichen, dass man Doctrine oder Propel verwenden kann. Ein klarer Pluspunkt für Propel ist, dass man die Entitäten und deren Relationen in einem einzigem File definieren kann. Weiters muss man sich bei Propel nicht um die Implementierung der Entitäten kümmern, da sie generiert werden. Außerdem liefert Propel zusätzlich die Möglichkeit aus einer bestehenden Datenbank die Entitäten zu "reverse-engineeren". Allerdings ist umsätzungstechnisch mit beiden Frameworks alles Mögliche möglich.