Sala 3: PHP

ORM: Doctrine

Doctrine 1 comenzó en 2006, pero hasta finales de 2010 no se lanzó **Doctrine 2**, una versión muy mejorada respecto a la anterior. Las **entidades en Doctrine 2** son **objetos PHP** que contienen **variables (propiedades)** que se guardan y devuelven a una base de datos a través del **Entity Manager de Doctrine**, una implementación del patrón de **mapeador de datos**. Su principal característica es la poca configuración que hace falta para empezar un proyecto.

Características

Una de las características de Doctrine es el bajo nivel de configuración que se necesita para comenzar un proyecto. Doctrine puede generar clases a partir de una base de datos creada, y el programador puede especificar relaciones y agregar funcionalidades comunes para las clases generadas. No hay necesidad de generar o mantener esquemas complejos en XML, como vistos en otros frameworks.

Otra característica de Doctrine es la habilidad de escribir consultas a la base de datos a partir de la programación orientada a objetos, llamada DQL (Doctrine Query Language). Estas clases proporcionan a los desarrolladores alternativas de gran alcance para SQL de mantener la flexibilidad y todavía permiten el cambio de base de datos de back-ends, sin necesidad de la duplicación de código.

Otras características

- Soporte para jerarquía (árbol de estructura) de datos.
- Soporte para los ganchos (métodos que puede validar o modificar la base de datos de entrada y salida) y los eventos a la estructura lógica de negocio relacionadas con ellos.
- Herencia columna de agregación (objetos similares se pueden almacenar en la tabla de base de datos, con un tipo de columna que especifica el subtipo del objeto en particular - la subclase correcta siempre se devuelve cuando se hace una consulta).
- Un marco de almacenamiento en caché, haciendo uso de varios backends como memcached, SQLite o APC.
- El ácido transacciones.
- Comportamientos del modelo (Sluggable, Timestampable, anidada conjunto, la internacionalización, registro de auditoría, el índice de búsqueda).
- Migraciones de bases de datos.

- Una "compilación" la función de combinar muchos archivos PHP del marco en una sola, para evitar el impacto en el rendimiento efectuados por lo general mediante la inclusión de los muchos archivos PHP de un marco.

El lector de anotaciones

Doctrine 2 introdujo las anotaciones para especificar la forma en que una clase PHP debía ser mapeada a una base de datos relacional (anteriormente sólo podían utilizarse archivos de configuración en Yaml, XML o PHP).

Las anotaciones no están escritas en PHP, se analizan mediante un lector de anotaciones desde una aplicación PHP, de esa forma afecta al funcionamiento de una aplicación. Son fáciles de memorizar y escribir, ni siquiera es necesario saber PHP. Es por ello que se considera un DSL (Domain-specific language) o lenguaje específico del dominio ya que permite convertir conceptos complejos en instancias sencillas.

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

/**
* @ORM\Entity
*/
class Post
{
    /**
    * @ORM\Column(type="string")
    */
    private $title;
}
```

Ejemplo de una entidad Producto

Las entidades suelen crearse por cada tabla de la base de datos. En este ejemplo se muestra como sería una tabla sencilla para productos:

```php
/**
 * @Entity @Table(name="productos")
 **/
class Producto
{
    /** @Id @Column(type="integer") @GeneratedValue **/
    protected $id;
    /** @Column(type="string") **/
    protected $nombre;

    public function getId()
    {
        return $this->id;
    }

    public function getNombre()
    {
        return $this->nombre;
    }

    public function setNombre($nombre)
    {
        $this->nombre = $nombre;
    }
}
```

Los mutadores getters y setters se definen para cada campo salvo para el campo *id*. Los mutadores permiten a Doctrine hacer llamadas que manipulan las entidades. El campo *id* no es un campo manipulable ya que es automático.

La anotación @*Entity* proporciona información acerca de la clase y el nombre de la tabla que representa. @*Id* con @*GeneratedValue* indican que es un valor automático de identidad de la base de datos, AUTO INCREMENT en el caso de MySql. @*Column* señala la existencia de una columna, junto con su argumento *type*, que indica el tipo de dato: *integer*, *string*...

Para que el servicio Entity Manager se entere de que una nueva entidad se debe introducir en la base de datos, se ha de llamar al método *persist()*. Para que finalmente esta entidad sea creada en la base de datos, se utiliza *flush()*.

```php
$newProductName = $argv[1];

$product = new Product();
$product->setName($newProductName);

$entityManager->persist($product);
$entityManager->flush();
```

```php
$productoRepository = $entityManager->getRepository('Producto');
$productos = $productoRepository->findAll();

foreach ($productos as $producto) {
    echo sprintf("-%s\n", $producto->getNombre());
}
```

El método *getRepository()* puede crear un objeto finder (llamado repositorio) para cada entidad. Contiene algunos métodos predefinidos como *findAll()*, pero se puede incluir cualquier método que se quiera para devolver resultados mediante sentencias SQL, DQL...

Para mostrar el nombre de un producto concreto basándose en su id, bastaría con el método *find()*:

```php
$id = $argv[1];
$producto = $entityManager->find('Producto', $id);

if ($producto === null) {
    echo "No se ha encontrado ningún producto.\n";
    exit(1);
}

echo sprintf("-%s\n", $producto->getNombre());
```

Para modificar un producto hacemos uso del mutador setNombre():

```php
$id = $argv[1];
$nuevoNombre = $argv[2];

$producto = $entityManager->find('Producto', $id);

if ($producto === null) {
    echo "El producto $id no existe.\n";
    exit(1);
}

$producto->setNombre($nuevoNombre);

$entityManager->flush();
```