

Data Objects Layer • PDO

Clase 10. Introducción

Carlos Ruiz Ruso • @micromante

Indice de contenidos

- **Introducción**
- **Requerimientos**
- **Constantes**
- **Conexiones y administración**
- **Transacciones**
- **Sentencias y almacenados**
- **Manejador de errores**
- **Clase PDO en profundidad**
- **Clase PDOStatement**
- **Ejercicio final sobre lo aprendido**

Problemas si no usamos PDO

- No existe consistencia entre las diferentes Bases de datos
- En ocasiones no existen los drivers
- Código duplicado
- Poco testeo del código final
- Alto mantenimiento

Solución...¿Qué es PDO?

- Objetos de datos de PHP (PDO en inglés)
- Capa de abstracción **de acceso a datos no de abstracción a la base de datos**
 - Cada base de datos implementa la interfaz PDO
 - Interfaz común para sistemas de bases de datos
- Escrito en C, muy rápido!
- Uso a partir de **PHP 5**

¿Porqué es necesario?

- Muchas extensiones de bases de datos nativas, diferentes interfaces de conexión. **UNIFICACIÓN**
- Se puede mantener retrocompatibilidad si se añaden nuevas funcionalidades a la interfaz. **ESCALABILIDAD**
- Múltiples soluciones de uso. **ADAPTABILIDAD**
 - Abstraction layers
 - Query builders
 - Data Access Objects

¿Qué puedo hacer con PDO?

- Tener un objeto de conexión a la BD
- Preparar y ejecutar una petición
- Enlazar objetos o variables
- Generar transacciones (lo veremos más adelante)
- Estados y control de errores

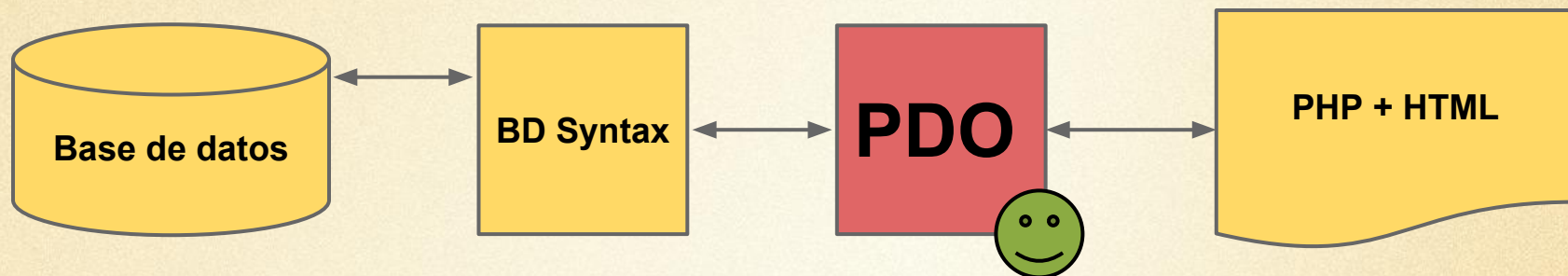
¿Qué bases de datos soporta?

- Actualmente PDO ofrece los siguientes drivers
 - **MySQL 3,4,5** [PDO_MYSQL]
 - PostgreSQL [PDO_PGSQL]
 - **SQLite 2,3** [PDO_SQLITE]
 - ODBC [PDO_ODBC]
 - Oracle [PDO_OCI]
 - Firebird [PDO_FIREBIRD]
 - ...

¿Qué tenemos entonces?

ABSTRACCIÓN · EFICIENCIA · UNIFICACIÓN

¿Cómo entenderlo facilmente?




¿Cómo instalar PDO?

- PDO viene con PHP 5.1
- Disponible como extensión PECL para PHP 5.0
- Requiere Orientación a objetos
- PDO se divide en 2 componentes
 - Core (Interfaz), activo por defecto
 - Driver (acceso a una base de datos particular)
 - Ejemplo: pdo_mysql

Extensión en PHP.INI

- Si abrimos php.ini o hacemos phpinfo();

PHP Version 5.3.5	
	
System	Windows NT CRUSH 6.1 build 7601 (Unknown Windows version Ultimate Edition Service Pack 1) i586
Build Date	Mar 9 2011 15:45:03
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscriphtolog configure ja --disable-pear --disable-ipv6 --disable-its --enable-cgi --disable-bcmath --disable-calendar --disable-odbc --disable-tokenizer --disable-xmlreader --disable-xmlwriter --without-sqlite --without-wddc --enable-debug-pack --enable-cli-win32 --enable-pdo --with-openssl --with-php-build --with-libxml --with-pdo-sqlite
Server API	CGIFastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	D:\Program Files\NetMake\VS\ZendServer\etc\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626.NTS.VC9
PHP Extension Build	API20090626.NTS.VC9



PDO

PDO support	enabled
PDO drivers	mysql, odbc, sqlite

pdo_mysql

PDO Driver for MySQL	enabled
Client API version	mysqlnd 5.0.7-dev - 091210 - \$Revision: 304625 \$

PDO_ODBC

PDO Driver for ODBC (Win32)	enabled
ODBC Connection Pooling	Enabled, strict matching

pdo_sqlite

PDO Driver for SQLite 3.x	enabled
SQLite Library	3.7.3

Pero yo uso MAMP, XAMPP...

- Normalmente los servidores suelen llevarlo ya de serie.
- En caso de no tenerlo, instalación
 - **Documentación oficial**
 - <http://www.php.net/manual/es/pdo.installation.php>

Una vez instalado y verificado

Entramos más en profundidad

CONSTANTES PREDEFINIDAS I

Estas **constantes** podemos verlas en **muchos frameworks**

- **PDO::FETCH_ASSOC**

- Devuelve cada fila como un array en función del nombre de las columnas.

- **PDO::FETCH_CLASS**

- Devuelve cada fila como un objeto en función del nombre de las columnas.

CONSTANTES PREDEFINIDAS II

Estas **constantes** podemos verlas en **muchos frameworks**

- **PDO::FETCH_UNIQUE**
 - Obtención solo de valores unicos.
- **PDO::FETCH_KEY_PAIR**
 - Devolver el resultado en array clave-valor .
- **PDO::FETCH_SERIALIZE**
 - Devolver como cadena serializada

CONSTANTES PREDEFINIDAS III

Estas constantes podemos verlas en muchos frameworks

- **PDO::ERRMODE_SILENT**
 - No emitir errores si ocurrieran
- **PDO::ERRMODE_EXCEPTION**
 - Lanzar una excepción tipo PDOException.
- **PDO::CASE_LOWER, CASE_UPPER**
 - Forzar mayúsculas o minúsculas en columnas

¿CÓMO ENCUENTRO LAS CONSTANTES?

En la documentación oficial de PDO puedes encontrar todas las constantes, en cada versión de PHP aparecen nuevas.

- <http://www.php.net/manual/es/pdo.constants.php>

CREAR UNA CONEXIÓN

Las conexiones se crean haciendo una instancia de la clase base PDO. El constructor acepta los parámetros de configuración del origen de los datos.

```
$dns = 'mysql:host=localhost,dbname=bdtest';
```

```
$con = new PDO($dns, $login, $password,$options);
```


OPCIONES DE CONEXIÓN

```
$con = new PDO($dns, $login, $password, $options);
```

El parámetro opciones acepta un array de la forma:

```
$options = array(PDO::ATTR_PERSISTENT => TRUE);
```

¿QUÉ PASA SI LA CONEXIÓN FALLA?

La conexión podría fallar por diferentes cosas, entre ellas que no tengamos bien los datos de conexión. Podemos recoger la excepción con PDOException.

Usamos por tanto TRY and Catch

EJEMPLO USO TRY

```
Try {  
    $db = new PDO(...);  
} catch(PDOException $event) {  
    echo 'Fallo por: ' . $event->getMessage();  
}
```

PERSISTENCIA EN LAS CONEXIONES

```
Try {  
    $options = array(PDO::ATTR_PERSISTENT => TRUE);  
    $db = new PDO("dsn", $user, $pass, $options);  
} catch(PDOException $event) {  
    echo 'Fallo por: ' . $event->getMessage();  
}
```


Let's go!

La ejecución de la Query en PDO puede hacerse de dos formas

- **Prepare o transacción**
 - Preparamos la query antes de lanzarla
 - Mejor seguridad y velocidad
- **Direct**
 - Lanzamos la query directamente al vuelo

Direct Query Execution

Cuando modificamos información necesitamos hacerlo con **exec()**

```
$db = new PDO(...);  
$db->exec("INSERT INTO users (email) VALUES('yo@gmail.  
com')");
```

Nos devolvera el numero de **filas afectadas** o **FALSE** si hay error

Debemos comprobar si devuelve 0 resultados o error en este caso.

Manejo de datos con PREPARE() o QUERY()

```
$db = new PDO(...);
```

```
$con = $db->prepare('...query syntax...');
```

//Podemos luego enlazar parámetros con **bindParam**, permite realizar plantillas de query y lanzarlas escapando los parametros para evitar ataques

```
$con->execute();
```

```
$db = new PDO(...);
```

```
$con = $db->query('...query syntax...');
```

```
//Lanzamos la query tal cual sobre la bd
```

Ejemplo

```
$db = new PDO(...);  
$con = $db->exec('...query syntax tipo inserción...');  
  
if($con !== false) {  
    echo 'Afectada ' . $con . ' ID insert ' . $db->lastInsertId();  
} else {  
    echo 'ERROR!';  
}
```


Direct Query Execution

Manejo de errores durante la ejecución

```
$db->errorInfo();
```

Nos devolvera algo similar a esto:

```
Array([0] => HYXXX, [1] => 1, [2] => Syntax error near... );
```

Mejor manera de manejar errores

La manera correcta es mediante las Exceptions de PHP

```
$db->setAttribute(  
    PDO::ATTR_ERRMODE,  
    PDO::ERRMODE_EXCEPTION  
);
```

Ahora el fallo lanzará una excepción que podremos recoger con php

Manejo de resultados

Quizás la flexibilidad más interesante de PDO es la flexibilidad a la hora de manejar los datos. Podemos manejarlos de la forma:

- **Array**, numeric
- String, columnas únicas o serializadas
- **Objects**
- Callback function...

Ejemplos de manejo de resultados

Array con indice numerico

```
$res = $db->query("SELECT * FROM users");  
  
while($row = $res->fetch(PDO::FETCH_NUM)) {  
    $row == array con claves numericas // [0 => 'a', 1 => 'b'...]  
}
```


Ejemplos de manejo de resultados

Array con índice asociado al nombre de la columna

```
$res = $db->query("SELECT * FROM users");

while($row = $res->fetch(PDO::FETCH_ASSOC)) {
    $row == array con claves asociadas // [email => 'a@a.com',
    nombre => 'juan'...]
}
```

Ejemplos de manejo de resultados

Array con índice numérico y asociado

```
$res = $db->query("SELECT * FROM users");
```

```
while($row = $res->fetch(PDO::FETCH_BOTH)) {  
    $row == array con claves asociadas // [email => 'a@a.com',  
    nombre => 'juan'...]  
}
```


Funciones Callback

El resultado de la operación puede ser enlazado a una funcion

```
function pinta_mensaje($texto) { echo $texto; }
```

```
$conn = $db->query('SELECT * .....');
```

```
$conn->fetchAll( PDO::FETCH_FUN, "pinta_mensaje");
```

Enlazado de parámetros

Las bases de datos más comunes aceptan el concepto de sentencia preparada.

Las ventajas son:

- La consulta solo necesita ser analizada una vez, pero puede ser ejecutada múltiples veces con los mismo o diferentes parámetros.
- Evitamos ciclo análisis/compilación/optimización
- No van entrecomillados los parámetros, sintaxis más clara
- **Previene SQL injection**

Ejemplo de enlazado de parámetros

...

```
$query = $conn->prepare("INSERT INTO users (name, value) VALUES (:name, :value);
```

```
$value = 'carlos';
```

```
$query->bindValue(':name', $value, PDO::PARAM_STR); //Añade el valor y ya no se puede  
cambiar el valor, cogeria el valor "carlos"
```

```
$query->bindParam(':value', $value, PDO::PARAM_STR); //Asociamos una variable que luego  
puede cambiar, en este caso cogeria el valor "luis"
```

```
$value = 'luis';
```

```
$query->execute();
```

...

Ejemplo de enlazado de parámetros reiterada

...

```
$query = $conn->prepare("INSERT INTO users (name, value) VALUES (?, ?);
```

```
$query->bindParam(1, 'Carlos');
```

```
$query->bindParam(2 'Test');
```

```
$query->execute(); //Tambien podemos ...execute(array('Carlos',Test'))
```

...

Ejemplo enlazado desde formulario

...

```
$value = $_POST['name'];
```

```
$query = $conn->prepare("SELECT * FROM users WHERE name = ?);
```

```
if($query->execute(array($value)) {
```

```
    while($fila = $query->fetch(PDO::FETCH_CLASS)) { print_r($fila); }
```

```
}
```

Los parámetros deberían comprobarse primero por seguridad

Objetos grandes (LOBs)

En el caso de necesitar almacenar datos grandes debemos utilizar la constante **PDO::PARAM_LOB**, esto permite almacenar binarios o imagenes.

```
$query->bindColumn(:file, $file, PDO::PARAM_LOB);
```

No es recomendable esta práctica pero debe conocerse que permite hacer este tipo de operaciones.

EJEMPLO BÁSICO DE CONEXIÓN A BD y LANZAMIENTO DE EXCEPCIONES

En las próximas clases aumentaremos el nivel de dificultad y aplicaremos estándares de desarrollo mediante clases

MUCHAS GRACIAS A TODOS!

Podeis seguirme en la redes sociales como @micromante o Carlos Ruiz Ruso
www.micromante.com