PHP Data Objects (PDO)

Camada de abstração de acesso a dados

Legenda

- ‘//’ e/ou Laranja: Comentários, observações, string e inteiros
- Verde: Palavras reservadas
- Azul: Classes, métodos e propriedades

Roteiro

- O que é?
- O que não é?
- Constantes pré-definidas
- Gerenciamento de conexões
- Transações de auto-commit
- Declarações preparadas
- Tratamento de erros
- LOBs
- Classe PDO, PDOStatement, PDOException
- PDO Drivers

O que é?

- Define uma interface leve e consistente para acessar bancos de dados em PHP.

O que não é?

- Não fornece uma abstração de banco de dados
- Não reescreve o SQL ou simula recursos faltantes

Constantes pré-definidas

- http://php.net/manual/pt_BR/pdo.constants.php

Gerenciamento de conexões

- Conexões são estabelecidas criando instancias da classe base PDO:

```
<?php
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user,
       $pass);
?>
```

Gerenciamento de conexões

- Exceções podem ser tratadas:

```
<?php
    Try{
       //conexão PDO
    } catch (PDOException $e){
       //mensagem da exceção (“$e->getMessage()”)
    }
?>
```

- Encerrando conexão

```
<?php
     $dbh = null;
<?
```

Gerenciamento de conexões

- Conexão persistentes
  - Não são fechadas ao final do script
  - São armazenadas em cache e reutilizadas quando outro script solicita uma conexão utilizando as mesmas credenciais.
  - Benefícios:
    - Evita sobrecarga
    - Resulta em uma aplicação mais rápida

Gerenciamento de conexões

- Conexões persistentes:

```
<?php
    $dbh = new PDO('mysql:host=localhost;dbname=test',
    $user, $pass, array(PDO::ATTR_PERSISTENT => true));
?>
```

Transações e auto-commit

- 4 principais recursos (ACID):
  - Atomicidade (Atomicity)
  - Consistência (Consistency)
  - Isolamento (Isolation)
  - Durabilidade (Durability)
- Finalidade:
  - Utilizadas para salvar um lote de mudanças a serem aplicadas de uma só vez

Transações e auto-commit

- Benefícios:
  - São realizadas com segurança
  - Não interferem em outras conexões
  - Podem ser desfeitos antes do commit
  - Facilitam tratamento de erros
- Malefícios:
  - Nem todos bancos de dados suportam transações
  - Necessário executar o PDO como “auto-commit” quando o banco de dados é aberto pela primeira vez

Transações e auto-commit

- Métodos:
  - beginTransaction() – Inicia a transação;
  - exec(“código sql”) – Incluir transações;
  - commit() – Conclui a transação;
  - rollBack() – Cancela a transação;

Transações e auto-commit

- Implementação

```
<?php
   try {
   //abre conexão
   $dbh->setAttribute(PDO::ATTR_ERRMODE,    PDO::ERRMODE_EXCEPTION);

   $dbh->beginTransaction();
   $dbh->exec("cod sql");
   $dbh->exec("cod sql");
   $dbh->commit();

   } catch(Exception $e){
   $dbh->rollBack();
   }
?>
```

Declarações preparadas

- Tipo de modelo compilado para o SQL que um aplicativo quer executar
- Podem ser personalizados utilizando parâmetros variáveis

Declarações preparadas

- Benefícios:
  - O código SQL é preparado apenas uma vez, mas pode ser executado várias vezes com os mesmos parâmetros ou diferentes
  - Evita repetir o ciclo de análise/compilação/otimização
  - Parâmetros não precisam ser citados
  - Evita injeção SQL
  - Facilitam reuso de código

Declarações preparadas

- Métodos:
  - prepare(“cod sql com os parâmetros”) – prepara a declaração
  - bindParam(“parâmetro”, “valor”) – parâmetros de ligação
  - execute() – executa a declaração preparada

Declarações preparadas

- Implementação:

```
<?php
    $stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
    $stmt->bindParam(":name", $name);
    $stmt->bindParam(":value", $value);

    //Inserindo uma informação (pode ser repetido n vezes)
    $name = 'one';
    $value = 1;
    $stmt->execute();
?>
```

Declarações preparadas

- Implementação (outras formas):

```
<?php
    $stmt = $dbh->prepare("INSERT INTO REGISTRY (name,
    value) VALUES (?, ?)");
    $stmt->bindParam(1, $name);
    $stmt->bindParam(2, $value);
?>
```

Declarações preparadas

- Implementação (Consulta):

```
<?php
   $stmt = $dbh->prepare("SELECT * FROM REGISTRY
   where name = ?");
   if(stmt->execute(array($_GET['name'])){
        while($row = $stmt->fetch()){
            print_r($row);
        }
   }
?>
```

Declarações preparadas

- Implementação (Consulta com parâmetro de entrada e saída):

```
<?php
   $stmt = $dbh->prepare("CALL
   sp_takes_string_returns_string(?)");
   $value = 'hello';
   //O valor de $value será utilizado para entrada e substituído
   pelo retorno
   $stmt->bindParam(1, $value, PDO::PARAM_STR|
   PDO::PARAM_INPUT_OUTPUT, 4000);

   $stmt->execute();
   print "procedure returned $value\n";
?>
```

Tratamento de erros

- PDO::ERRMODE_SILENT
  - Modo padrão. PDO configurará o código de erro para ser inspecionado utilizando os métodos errorCode() e errorInfo()
- PDO::ERRMODE_WARNING
  - Além de definir o erro, o PDO emitirá uma mensagem E_WARNING. Configuração útil para verificar os erros sem interromper o fluxo do aplicativo
- PDO::ERRMODE_EXCEPTION
  - Configura e lança o PDOException, permitindo que os erros sejam tratados

Tratamento de erros

- PDO::ERRMODE_EXCEPTION

```
<?php
    try {
        $dbh = new PDO($dsn, $user, $password);
        $dbh->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    } catch (PDOException $e) {
        echo 'Connection failed: ' . $e->getMessage();
    }
?>
```

Manipulando objetos grandes (LOBs)

- Large Objects (LOBs)
- Permite armazenar dados grandes no banco de dados (4kb ou mais)
- Utilização do PDO::PARAM_LOB nas chamadas:
  - bindParam()
  - binColumn()
- PARAM_LOB informa ao PDO para mapear os dados como um fluxo

Manipulando objetos grandes (LOBs)

- Após o LOB ser representado como fluxo, é possível a utilização dos métodos:
  - fgets()
  - fread()
  - stream_get_contents()

Manipulando objetos grandes (LOBs)

- Este exemplo liga o LOB a variável $lob, depois enviada para o navegador utilizando fpassthru().
- Implementação:

```
<?php
//conexão DB
$stmt = $db->prepare("select ... From ... Where    id=?");
$stmt->execute(array($_GET['id']));
$stmt->bindColumn(1, $type, PDO::PARAM_STR,  256);
$stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
$stmt->fetch(PDO::FETCH_BOUND);
header("Content-type: $type");
fpassthru($lob);
?>
```

Sinopse da classe PDO

```
PDO {
   public __construct ( string $dsn [, string $username [, string $password [, array $options ]]] )
   public bool beginTransaction ( void )
   public bool commit ( void )
   public mixed errorCode ( void )
   public array errorInfo ( void )
   public int exec ( string $statement )
   public mixed getAttribute ( int $attribute )
   public static array getAvailableDrivers ( void )
   public bool inTransaction ( void )
   public string lastInsertId ([ string $name = NULL ] )
   public PDOStatement prepare ( string $statement [, array $driver_options = array() ] )
   public PDOStatement query ( string $statement )
   public string quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] )
   public bool rollBack ( void )
   public bool setAttribute ( int $attribute , mixed $value )
}
```

Sinopse da classe PDO índice

- PDO::beginTransaction — Initiates a transaction
- PDO::commit — Commits a transaction
- PDO::__construct — Creates a PDO instance representing a connection to a database
- PDO::errorCode — Fetch the SQLSTATE associated with the last operation on the database handle
- PDO::errorInfo — Fetch extended error information associated with the last operation on the database handle
- PDO::exec — Execute an SQL statement and return the number of affected rows
- PDO::getAttribute — Recuperar um atributo da conexão com o banco de dados
- PDO::getAvailableDrivers — Retorna um array com os drivers PDO disponíveis
- PDO::inTransaction — Checks if inside a transaction
- PDO::lastInsertId — Returns the ID of the last inserted row or sequence value
- PDO::prepare — Prepares a statement for execution and returns a statement object
- PDO::query — Executes an SQL statement, returning a result set as a PDOStatement object
- PDO::quote — Quotes a string for use in a query.
- PDO::rollBack — Rolls back a transaction
- PDO::setAttribute — Set an attribute

Sinopse da classe PDOStatement

```
PDOStatement implements Traversable {

    /* Propriedades */
    readonly string $queryString;

    /* Métodos */
    public bool bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen [, mixed $driverdata ]]] )
    public bool bindParam ( mixed $parameter , mixed &$variable [, int $data_type = PDO::PARAM_STR [, int $length [,
    mixed $driver_options ]]] )
    public bool bindValue ( mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR ] )
    public bool closeCursor ( void )
    public int columnCount ( void )
    public void debugDumpParams ( void )
    public string errorCode ( void )
    public array errorInfo ( void )
    public bool execute ([ array $input_parameters ] )
    public mixed fetch ([ int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0
    ]]] )
    public array fetchAll ([ int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array() ]]] )
    public mixed fetchColumn ([ int $column_number = 0 ] )
    public mixed fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ]] )
    public mixed getAttribute ( int $attribute )
    public array getColumnMeta ( int $column )
    public bool nextRowset ( void )
    public int rowCount ( void )
    public bool setAttribute ( int $attribute , mixed $value )
    public bool setFetchMode ( int $mode )
}
```

Sinopse da classe PDOStatement índice

- PDOStatement::bindColumn — Bind a column to a PHP variable
- PDOStatement::bindParam — Binds a parameter to the specified variable name
- PDOStatement::bindValue — Binds a value to a parameter
- PDOStatement::closeCursor — Closes the cursor, enabling the statement to be executed again.
- PDOStatement::columnCount — Returns the number of columns in the result set
- PDOStatement::debugDumpParams — Dump an SQL prepared command
- PDOStatement::errorCode — Fetch the SQLSTATE associated with the last operation on the statement handle
- PDOStatement::errorInfo — Fetch extended error information associated with the last operation on the statement handle
- PDOStatement::execute — Executes a prepared statement
- PDOStatement::fetch — Fetches the next row from a result set
- PDOStatement::fetchAll — Returns an array containing all of the result set rows
- PDOStatement::fetchColumn — Returns a single column from the next row of a result set
- PDOStatement::fetchObject — Fetches the next row and returns it as an object.
- PDOStatement::getAttribute — Retrieve a statement attribute
- PDOStatement::getColumnMeta — Returns metadata for a column in a result set
- PDOStatement::nextRowset — Advances to the next rowset in a multi-rowset statement handle
- PDOStatement::rowCount — Returns the number of rows affected by the last SQL statement
- PDOStatement::setAttribute — Define um atributo na instrução
- PDOStatement::setFetchMode — Define o modo de carga de dados para esta instrução

Sinopse da classe PDOException

```
PDOException extends RuntimeException {

    /* Propriedades */
    public array $errorInfo ;
    protected string $code ;

    /* Propriedades herdadas */
    protected string $message ;
    protected int $code ;
    protected string $file ;
    protected int $line ;

    /* Métodos herdados */
    final public string Exception::getMessage ( void )
    final public Exception Exception::getPrevious ( void )
    final public mixed Exception::getCode ( void )
    final public string Exception::getFile ( void )
    final public string Exception::getLine ( void )
    final public array Exception::getTrace ( void )
    final public string Exception::getTraceAsString ( void )
    public string Exception::__toString ( void )
    final private string Exception::__clone ( void )
}
```

PDO Drivers

- CUBRID (PDO)
- Firebird (PDO)
- IBM (PDO)
- Informix (PDO)
- MySQL (PDO)
- MS SQL Server (PDO)
- Oracle (PDO)
- ODBC and DB2 (PDO)
- PostgreSQL (PDO)
- SQLite (PDO)
- 4D (PDO)

Referências

- http://php.net/manual/pt_BR/book.pdo.php