PHP

February 20, 2024

Mécanisme d'exception

Les exceptions

Principe :

- Condition particulière sujette à un traitement particulier
- Exemple : absence d'un fichier utile, impossibilité de se connecter à une base de données, mémoire saturée, débordement d'indice de tableau, division par zéro.
- Les exceptions ne traitent pas les bugs mais elles traitent des "erreurs d'implémentation que l'on peut prévoir".
- Exemple d'exceptions imprévisibles : erreur http, pas de réponse d'un appel bdd, pas de réponse (ou timeout) d'un appel API externe.....
- En cas d'exception le déroulement normal du programme est interrompue pour traitement de l'exception.
- Le mécanisme d'exception
  - Émission d'une exception via le mot-clé : **throw**
  - Capture et traitement de l'exception : **try... catch()...**

Les exceptions

Principe :

- Condition particulière sujette à un traitement particulier
- Exemple : absence d'un fichier utile, impossibilité de se connecter à une base de données, mémoire saturée, débordement d'indice de tableau, division par zéro.
- Les exceptions ne traitent pas les bugs mais elles traitent des "erreurs d'implémentation que l'on peut prévoir".
- Exemple d'exceptions imprévisibles : erreur http, pas de réponse d'un appel bdd, pas de réponse (ou timeout) d'un appel API externe.....
- En cas d'exception le déroulement normal du programme est interrompue pour traitement de l'exception.
- Le mécanisme d'exception
  - Émission d'une exception via le mot-clé : **throw**
  - Capture et traitement de l'exception : **try... catch()...**

Les exceptions

Principe :

- Condition particulière sujette à un traitement particulier
- Exemple : absence d'un fichier utile, impossibilité de se connecter à une base de données, mémoire saturée, débordement d'indice de tableau, division par zéro.
- Les exceptions ne traitent pas les bugs mais elles traitent des "erreurs d'implémentation que l'on peut prévoir".
- Exemple d'exceptions imprévisibles : erreur http, pas de réponse d'un appel bdd, pas de réponse (ou timeout) d'un appel API externe.....
- En cas d'exception le déroulement normal du programme est interrompue pour traitement de l'exception.
- Le mécanisme d'exception
  - Émission d'une exception via le mot-clé : **throw**
  - Capture et traitement de l'exception : **try... catch()...**

Les exceptions

Principe :

- Condition particulière sujette à un traitement particulier
- Exemple : absence d'un fichier utile, impossibilité de se connecter à une base de données, mémoire saturée, débordement d'indice de tableau, division par zéro.
- Les exceptions ne traitent pas les bugs mais elles traitent des "erreurs d'implémentation que l'on peut prévoir".
- Exemple d'exceptions imprévisibles : erreur http, pas de réponse d'un appel bdd, pas de réponse (ou timeout) d'un appel API externe.....
- En cas d'exception le déroulement normal du programme est interrompue pour traitement de l'exception.
- Le mécanisme d'exception
  - Émission d'une exception via le mot-clé : **throw**
  - Capture et traitement de l'exception : **try... catch()...**

Les exceptions

Principe :

- Condition particulière sujette à un traitement particulier
- Exemple : absence d'un fichier utile, impossibilité de se connecter à une base de données, mémoire saturée, débordement d'indice de tableau, division par zéro.
- Les exceptions ne traitent pas les bugs mais elles traitent des "erreurs d'implémentation que l'on peut prévoir".
- Exemple d'exceptions imprévisibles : erreur http, pas de réponse d'un appel bdd, pas de réponse (ou timeout) d'un appel API externe.....
- En cas d'exception le déroulement normal du programme est interrompue pour traitement de l'exception.
- Le mécanisme d'exception
  - Émission d'une exception via le mot-clé : **throw**
  - Capture et traitement de l'exception : **try... catch()...**

Exemple exception

- génération d'une exception (mot clé **throw**)

```
class MonObjet {
function test($n=null) {
if ($n==null) throw new Exception("La valeur ne
    convient pas");
return true;
}}
```

Exemple exception

- capture de l'exception et traitement
  try ... catch () ... .

```php
$obj = new MonObjet();
try {
// utilisation d'une méthode susceptible
// d'émettre une exception
$obj->test();
}
catch (Exception $e) {
// traitement de l'exception
print($e->getMessage());
}
```

- Une exception est une instance de la classe **Exception**.
- Elle possède un ensemble de méthodes utiles pour le débogage
  - **getMessage()** : Retourne la valeur du message
  - **getCode()** : Retourne la valeur du code d'erreur
  - **getFile()** : Retourne le nom du fichier d'où l'exception a été émise
  - **getLine()** : Retourne le numéro de la ligne où l'exception a été émise
  - **getTrace()** : Retourne un tableau de la trace d'erreur
  - **getTraceAsString()**: Retourne une chaîne de caractères correspondant à la trace de l'erreur

Créer vos propres classes d'exceptions : La classe créer étendra la classe Exception en précisant le message d'erreur, et permet de gérer les erreurs spécifiquement en fonction de l'origine

- Meilleure gestion des exceptions
- Personnalisation du traitement suivant le type d'exception

```php
<?php
class customException extends Exception {
  public function errorMessage() {
    $errorMsg = 'Error on line '.$this->getLine().' in '.
        $this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid';
    return $errorMsg;
  }
}
$email = "someone@example...com";
try {
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
      {
    //throw exception if email is not valid
    throw new customException($email);
  }
}
catch (customException $e) {
  echo $e->errorMessage();
}
?>
```

PDO

PDO = PHP Data Object

Permet d'utiliser les bases de données suivantes :

- DB2
- MySQL
- MySQLI
- SQLLite
- ODBC
- Oracle
- PostGreSQL

Attention

if faut que l'extension PDO soit activée dans PHP.ini (vérifier avec phpinfo() l'activation de PDO et de l'extension de votre base de donnée)

Principe

- On définit le type de base de données utilisée à la connexion
- PDO utilise les fonctions d'accès appropriées
- PDO fournit une interface d'abstraction à l'accès de données,
- **Permet de changer de serveur de BD en modifiant uniquement le script de connexion**
- Attention : il ne réécrit pas le SQL, il faut utiliser du SQL standard!
- L'accès à la BD est géré via un objet PDO

Connexion

Exemple

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

Recommandation

Utilisez des constantes définies dans un fichier xxx.inc.php

Gestion de la connexion

Remarque

- la connexion est active tant que l'objet PDO l'est (objet retourné si succès de la connexion)
- Pour clore la connexion :
  - détruire l'objet en s'assurant que toutes ses références sont effacées
  - assigner NULL à la variable gérant l'objet
  - Sinon, PHP fermera automatiquement la connexion à la fin du script.

```php
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user,
    $pass);
// utiliser la connexion ici
$dbh = null;// et maintenant, fermez-la !
?>
```

Gestion de la connexion

Remarque

- la connexion est active tant que l'objet PDO l'est (objet retourné si succès de la connexion)
- Pour clore la connexion :
  - détruire l'objet en s'assurant que toutes ses références sont effacées
  - assigner NULL à la variable gérant l'objet
  - Sinon, PHP fermera automatiquement la connexion à la fin du script.

```php
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user,
    $pass);
// utiliser la connexion ici
$dbh = null;// et maintenant, fermez-la !
?>
```

Connexions persistantes :

- elles ne sont pas fermées à la fin du script
- elles sont mises en cache
- elles sont réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres (hôte, utilisateur, mot de passe)

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

Remarques ;

Les connexions persistantes sont un bon moyen d'accélérer les accès à une base SQL : économie du temps de 'reconnexion".

Connexions persistantes :

- elles ne sont pas fermées à la fin du script
- elles sont mises en cache
- elles sont réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres (hôte, utilisateur, mot de passe)

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

Remarques ;

Les connexions persistantes sont un bon moyen d'accélérer les accès à une base SQL : économie du temps de 'reconnexion".

Connexions persistantes :

- elles ne sont pas fermées à la fin du script
- elles sont mises en cache
- elles sont réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres (hôte, utilisateur, mot de passe)

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

Remarques ;

Les connexions persistantes sont un bon moyen d'accélérer les accès à une base SQL : économie du temps de 'reconnexion".

Connexions persistantes :

- elles ne sont pas fermées à la fin du script
- elles sont mises en cache
- elles sont réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres (hôte, utilisateur, mot de passe)

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

Remarques ;

Les connexions persistantes sont un bon moyen d'accélérer les accès à une base SQL : économie du temps de 'reconnexion".

Connexions persistantes :

- elles ne sont pas fermées à la fin du script
- elles sont mises en cache
- elles sont réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres (hôte, utilisateur, mot de passe)

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

Remarques ;

Les connexions persistantes sont un bon moyen d'accélérer les accès à une base SQL : économie du temps de 'reconnexion".

Test de la connexion

Attraper l'exception lancée par le constructeur

```php
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test',
        $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Erreur !: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Exécuter une requête

Deux possibilités :

- **PDO::exec()** : Exécute une requête SQL
  retourne le nombre de lignes affectées
  - Attention : ne retourne pas de résultat pour une requête SELECT.
  - à utiliser pour les requêtes de type INSERT, DELETE,...
- **PDO::query()** : Exécute une requête SQL
  retourne un jeu de résultats en tant qu'objet PDOStatement
  - PDOStatement est souvent inutile pour les requêtes de type INSERT, DELETE,..⇒ utiliser exec.
  - à utiliser pour les requêtes de type SELECT

Exemple EXEC

```php
<?php
$dbh = new PDO('odbc:sample', 'db2inst1', 'ibmdb2');
/* Effacement de toutes les lignes de la table FRUIT */
$count = $dbh->exec("DELETE FROM fruit WHERE couleur = '
    rouge'");

/* Retourne le nombre de lignes effacées */
print("Retourne le nombre de lignes effacées :\n");
print("Effacement de $count lignes.\n");
?>
```

Exemple QUERY

```php
<?php
function getFruit($conn) {
    $sql = 'SELECT name, color, calories FROM fruit
        ORDER BY name';
    foreach ($conn->query($sql) as $row) {
        print $row['name'] . "\t";
        print $row['color'] . "\t";
        print $row['calories'] . "\n";
  }
}
?>
```

Traitement du résultat de QUERY

PDO::query(..)

- retourne un objet PDOStatement (jeu de résultats)
- l'interface '**iterator**' permet de parcourir le jeu de résultats

```php
foreach ($conn->query($sql) as $row) {
    print $row['name'] . "\t";
```

- Chaque ligne du résultat sera affectée successivement au tableau associatif **$row**
- les clés du tableau sont les champs retournés par la requête.

Traitement du résultat de QUERY

PDO::fetch()

```
$resultats=$connexion->query("SELECT membre FROM membres
    WHERE champ_id_membre='id_membre'");
 while( $resultat = $resultats->fetch(PDO::FETCH_OBJ) )
{
        echo 'Utilisateur : '.$resultat->membre.'<br>';
}
$resultats->closeCursor(); //Ferme le curseur, permettant
    à la requête d'être de nouveau exécutée
```

Les options de PDO::fetch(OPTION)

PDO::FETCH_ASSOC	retourne un tableau indexé par le nom de la colonne
PDO::FETCH_NUM	retourne un tableau indexé par le numéro de la colonne
PDO::FETCH_BOTH (défaut)	retourne un tableau indexé par les noms de colonnes et par les numéros de colonnes
PDO::FETCH_NAMED	retourne un tableau de la même forme que PDO::FETCH_ASSOC, excepté que s'il y a plusieurs colonnes avec les mêmes noms, la valeur pointée par cette clé sera un tableau de toutes les valeurs de la ligne qui a ce nom comme colonne
PDO::FETCH_OBJ	retourne un objet anonyme avec les noms de propriétés qui correspondent aux noms des colonnes
PDO::FETCH_CLASS	retourne une nouvelle instance de la classe demandée, liant les colonnes du jeu de résultats aux noms des propriétés de la classe.

FetchMode pour foreach

Remarque

- Les options de PDO::fetch() peuvent être utilisées pour le **foreach**
- en utilisant la méthode **setFetchMode(OPTION)**

```
$res=$conn->query($sql) ;
$res->setFetchMode( PDO::FETCH_OBJ) ;
   foreach  ( $res as $obj) {
        echo $obj->name . "\t";
```

Transactions et validation automatique

- PDO gère les transactions avant d'exécuter des requêtes.
- N'importe quel '**travail mené à bien**' dans une transaction
  - est garanti d'être appliqué à la base de données sans risque
  - sans interférence pour les autres connexions (si validée).
- Mode autocommit (car toutes les BD ne gère pas les transactions)
  - chaque requête a sa transaction implicite si le SGBG le supporte
  - aucune transaction si le SGDG le supporte pas.

Transactions et validation automatique

- PDO gère les transactions avant d'exécuter des requêtes.
- N'importe quel '**travail mené à bien**' dans une transaction
  - est garanti d'être appliqué à la base de données sans risque
  - sans interférence pour les autres connexions (si validée).
- Mode autocommit (car toutes les BD ne gère pas les transactions)
  - chaque requête a sa transaction implicite si le SGBG le supporte
  - aucune transaction si le SGDG le supporte pas.

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Utiliser les transactions

L'intérêt de gérer les transactions est de garantir une unicité et une fiabilité des données stockées en bdd.
ACID : acronyme désignant les termes : Atomicité, Cohérence, Isolation et Durabilité.

- Atomicité : tout changement effectué doit être accompli jusqu'au bout
- Cohérence : respecte contraintes d'intégrité des données
- Isolation : sérialisable, pas de transation simultanée si risque d'impact
- Durabilité : tout changement en bdd est permanent

Pour utiliser les transactions

Démarrer une transaction : PDO::beginTransaction()
Valider une transaction : PDO::commit()
Annuler toute la transaction : PDO::rollBack()

Exemple de transaction avec gestion des exceptions

```
try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::
      ERRMODE_EXCEPTION);
// activation du rapport d'erreurs et des exceptions
  $dbh->beginTransaction();
  $dbh->exec("insert into staff (id, first, last) values
      (23, 'Joe', 'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount,
      changedate)
       values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

Gestion des requêtes préparées

Requête préparée = "requête compilée et paramétrée"

- elles sont "préparées" (i.e. analysée une seule fois analyse/compilation/optimisation)
- elles peuvent être exécutées plusieurs fois avec des paramètres différents
- gain de temps pour des requêtes complexes

Remarque

- Les requêtes préparées sont tellement pratiques que c'est l'unique fonctionnalité que PDO émule pour les drivers qui ne les supportent pas.
- rend le code indépendant des SGBD

Gestion des requêtes préparées

Requête préparée = "requête compilée et paramétrée"

- elles sont "préparées" (i.e. analysée une seule fois analyse/compilation/optimisation)
- elles peuvent être exécutées plusieurs fois avec des paramètres différents
- gain de temps pour des requêtes complexes

Remarque

- Les requêtes préparées sont tellement pratiques que c'est l'unique fonctionnalité que PDO émule pour les drivers qui ne les supportent pas.
- rend le code indépendant des SGBD

Exemple de requête préparée

Insertions répétées en utilisant des requêtes réparées

```
<?php
//On définit la requête préparée avec des paramètres
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();
// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Exemple de requête préparée

Insertions répétées en utilisant des requêtes réparées

```php
<?php
//On définit la requête préparée avec des paramètres
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();
// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Exemple de requête préparée

Insertions répétées en utilisant des requêtes réparées

```
<?php
//On définit la requête préparée avec des paramètres
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();
// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Exemple de requête préparée

Insertions répétées en utilisant des requêtes réparées

```
<?php
//On définit la requête préparée avec des paramètres
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value)
    VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();
// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Définir les paramètres d'une procédure préparée

- Soit par '**:nom_du_parametre**'

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name,
    value) VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables via le nom du parametre
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
```

- Soit par marqueurs (i.e le caractère '?').

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name,
    value) VALUES (?, ?)");
//On lie les paramètres de la requête aux variables via le num d'apparition
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);
```

Définir les paramètres d'une procédure préparée

- Soit par '**:nom_du_parametre**'

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name,
    value) VALUES (:name, :value)");
//On lie les paramètres de la requête aux variables via le nom du parametre
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);
```

- Soit par marqueurs (i.e le caractère '?').

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name,
    value) VALUES (?, ?)");
//On lie les paramètres de la requête aux variables via le num d'apparition
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);
```

Associer des variables aux paramètres

PDOStatement::bindParam — Lie un paramètre à un nom de variable spécifique

```php
<?php
/* Exécution d'une requête préparée en liant des
   variables PHP */
$calories = 150;
$couleur = 'rouge';
$sth = $dbh->prepare('SELECT nom, couleur, calories
    FROM fruit
    WHERE calories < :calories AND couleur = :couleur');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':couleur', $couleur, PDO::PARAM_STR, 12)
    ;
$sth->execute();
?>
```

Associer des valeurs aux paramètres

PDOStatement::bindValue — Associe une valeur à un paramètre

```php
<?php
/* Exécute une requête préparée en associant des
   variables PHP */
$calories = 150;
$couleur = 'rouge';
$sth = $dbh->prepare('SELECT nom, couleur, calories
    FROM fruit
    WHERE calories < :calories AND couleur = :couleur');
$sth->bindValue(':calories', $calories, PDO::PARAM_INT);
$sth->bindValue(':couleur', $couleur, PDO::PARAM_STR);
$sth->execute();
?>
```

Constantes prédéfinies

PDO utilise des constantes prédéfinies :

PDO::PARAM_BOOL	Représente le type de données booléen.
PDO::PARAM_NULL	Représente le type de données NULL SQL.
PDO::PARAM_INT	Représente le type de données INTEGER SQL.
PDO::PARAM_STR	Représente les types de données CHAR, VARCHAR ou les autres types de données sous forme de chaîne de caractères SQL.
PDO::PARAM_LOB	Représente le type de données "objet large" SQL.

bindParam vs bindValue

bindParam

```php
$sex = 'male';
$s = $dbh->prepare('SELECT name FROM students WHERE sex =
    :sex');
$s->bindParam(':sex', $sex); // use bindParam to bind the
    variable
$sex = 'female';
$s->execute(); // executed with WHERE sex = 'female'
```

bindValue

```php
$sex = 'male';
$s = $dbh->prepare('SELECT name FROM students WHERE sex =
    :sex');
$s->bindValue(':sex', $sex); // use bindValue to bind the
    variable's value
$sex = 'female';
$s->execute(); // executed with WHERE sex = 'male'
```

bindParam vs bindValue

bindParam

```
$sex = 'male';
$s = $dbh->prepare('SELECT name FROM students WHERE sex =
    :sex');
$s->bindParam(':sex', $sex); // use bindParam to bind the
    variable
$sex = 'female';
$s->execute(); // executed with WHERE sex = 'female'
```

bindValue

```
$sex = 'male';
$s = $dbh->prepare('SELECT name FROM students WHERE sex =
    :sex');
$s->bindValue(':sex', $sex); // use bindValue to bind the
    variable's value
$sex = 'female';
$s->execute(); // executed with WHERE sex = 'male'
```

Exemple de requête préparée

Récupération des données en utilisant des requêtes préparées

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