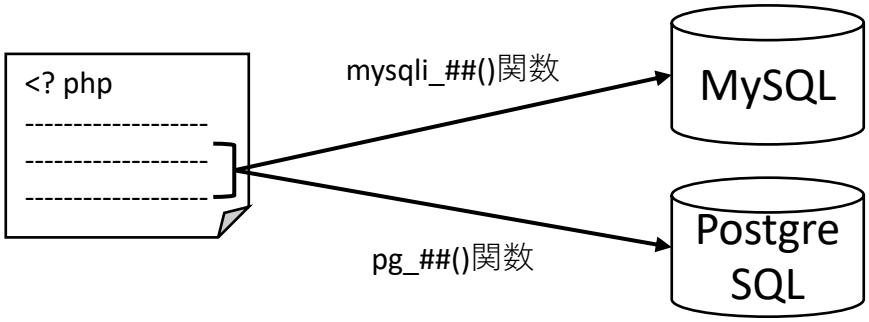
Lesson 2 PHP-DBの復習

■お品書き

1. PHP-DB今昔物語
2. PDOによるDB接続
3. 例外処理
4. データ処理
5. データ取得のTIPS

[1] PHP-DB今昔物語

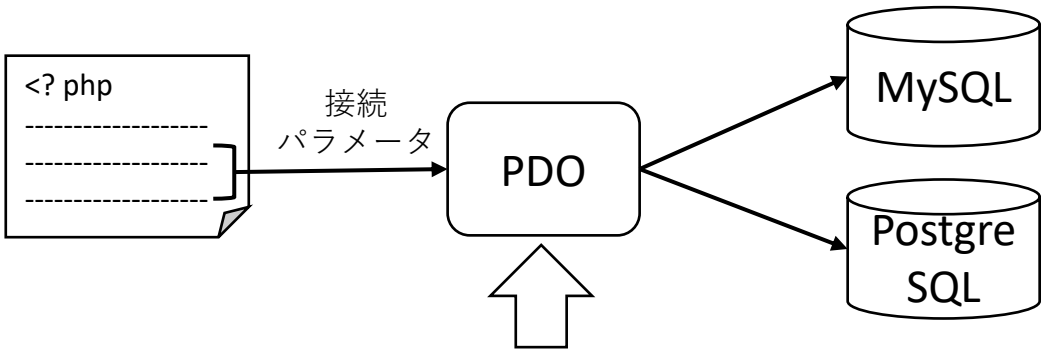
(1) 昔



接続先DBによって関数が違っていた。

接続先DBの種類が変更になったら、PHPのソースコードのいたるところを変更しないといけなかった。

(2) 今



DB接続を抽象化

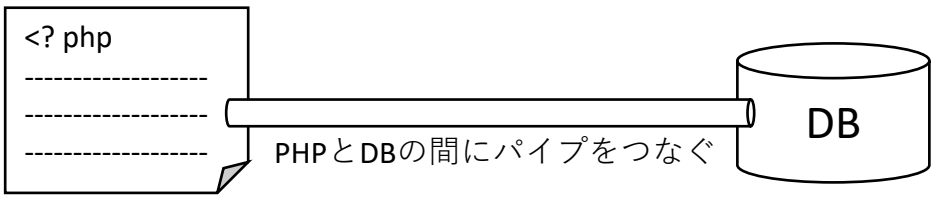
PDOクラスを使うことでDB接続を抽象化できる。
接続先DBを変更したければPDOに渡す接続パラメータを変更するだけでよい!

[2] PDOによるDB接続

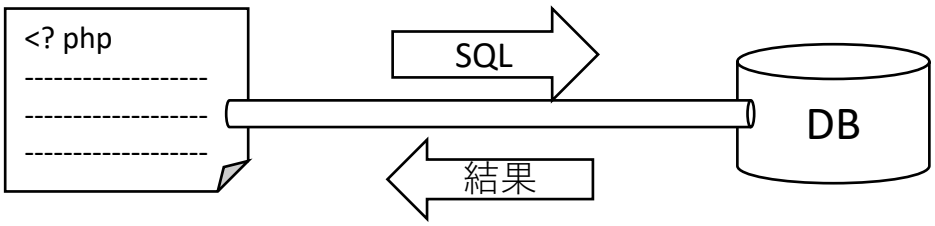
(1) 処理概説

PHPのPDOを使ってのDB処理の基本手順は、以下の3ステップで行う。

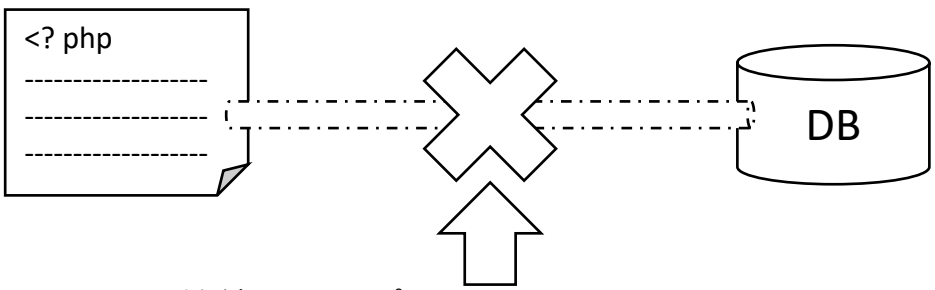
1. 接続



2. データ処理



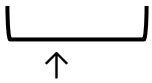
3. 切断



DB接続のパイプはリソースを必要とするので、確実に切断しておく必要がある。

(2) PDOでの記述

PDOでの手順は以下の通り。

1. 接続
   PDOインスタンスを生成。
   $db = new PDO(… , … , …);
   
   3コの引数については後述
2. データ処理
   お品[4]の解説をお楽しみに。
3. 切断
   PDOインスタンスの破棄。
   $db = null;

(3) PDOコンストラクタの引数

PDOコンストラクタ(new時)の引数は以下の3コ。

1. $dsn: データソース名。
   DB接続用の文字列。接続先DBごとに書式が異なる。MySQLは以下の通り。
   mysql:host=ホストアドレス;dbname=データベース名;charset=utf8
   ↑
   オマジナイ
2. $username: ユーザ名。
3. $password: パスワード。

(注)
上記3コの引数の値は直接記述するのではなく、どこかに変数か定数としてまとめて記述しておく。すると、接続先DBを変更するとき、その1ヶ所を変更するだけですむ。

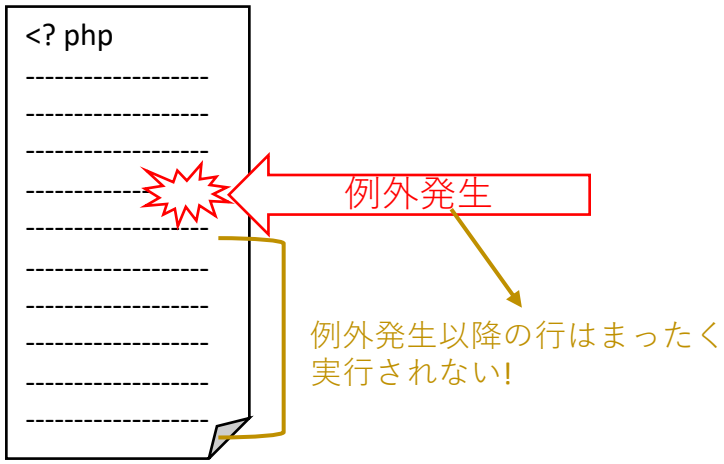
[3] 例外処理

(1) エラーと例外

オブジェクト指向プログラミングでは、古くからあるエラーとは別に例外という考え方が取り入れられている。違いは以下の通り。

- エラー(Error)
  発生したら対処不能。発生しないようにコーディングする。
- 例外(Exception)
  発生したら、それをうけとって対処可能。
  逆に、対応不能なものは例外として発生させてはダメ。

(2) 例外発生

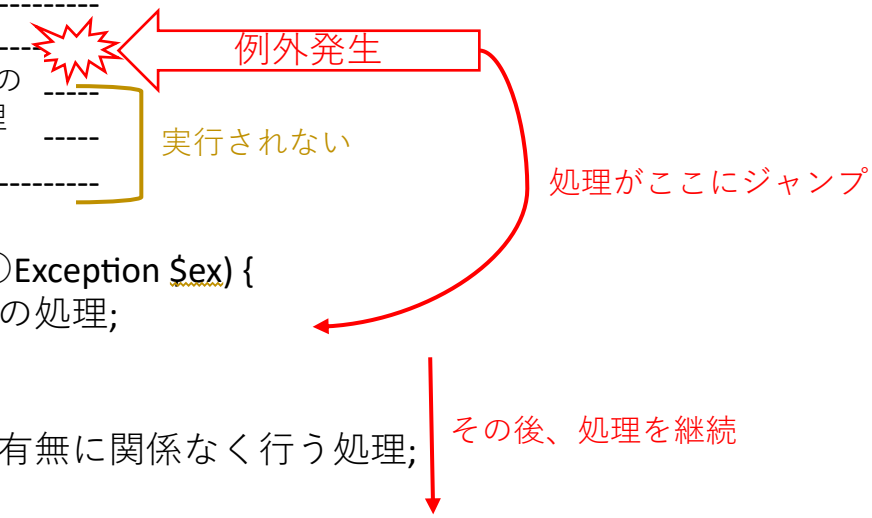


オブジェクト指向言語では、さまざまなところで例外が発生するように組み込まれているが、例外発生以降のコードはまったく実行されなくなってしまう!
うまく対処してプログラミング(例外処理)していく必要がある。

(3) try-catch-finally

例外処理は以下の構文。

```
try {
    ----------
    ----------
    --  通常の  -----
        処理
    --          -----
    ----------
}
catch(〇〇Exception $ex) {
    例外時の処理;
}
finally {
    例外の有無に関係なく行う処理;
}
```



- ◆ catchブロックの( )内に記述する「〇〇Exception」は発生する例外クラスを記述(マニュアルに記載されている)。
  - ● その変数$exに実際に発生した例外インスタンスが格納されている。
  - ● $exのメソッドをいろいろ実行することで例外の内容を確認できる(後述)。
- ◆ catchブロックは必要に応じていくつも重ねることができる。

(4) 例外クラスのメソッド

例外クラスのメソッドには以下のようなものがある。

- getCode(): 例外コードを取得。
- getFile(): 例外が発生したファイルを取得。
- getLine(): 例外が発生した行を取得。
- getMessage(): 例外メッセージを取得。
- getTraceAsString():
  例外が発生するまでの経緯(スタックトレース)を取得。

(5) PDOと例外処理

DB接続(PDOの利用)は例外発生がつき物なので、確実に例外処理を行う。さらに、例外が発生してもDB接続を切断できるようにしておく必要もある。

以下のソースコードパターンとなる。

```
try {
    $db = new PDO(…);
        :
    以降の処理
        :
}
catch(PDOException $ex) {
    例外処理(通常はログへの記述処理)
}
finally {
    $db = null;
}
```

[4] データ処理

(1) データ処理の基本

PDOでのデータ処理は、PDOStatementオブジェクトが中心的役割となる。DB接続と切断の間に以下の手順で行う。

1. SQL文字列を作成する。
   普通の文字列として作成(SQL文末のセミコロンは不要)。
   $sql = “INSERT …”;
2. PDOStatementオブジェクトを取得する。
   PDOオブジェクトから取得する。引数として1の文字列をわたす。
   $stmt = $db->prepare($sql);
   （PDOStatement：$stmt、PDO：$db）
3. 変数のバインドを行う。
   →後述。
4. 実行する。
   PDOStatementオブジェクトに対して実行する。
   戻り値は実行が成功した(true)か失敗した(false)かを表す。
   $result = $stmt ->execute();

1で作成するSQLは、あらかじめコマンドなどで実行し、動くことが確認できているものをコピペする。

(2) バインド変数

たとえば、

DELETE FROM orders WHERE order_id = 8000;

というSQLの「8000」の部分は入力値などで変動する。
これを変数として、

$orderId = 8000;

とする。すると、手順1で作成するSQ文字列は

$sql = “DELETE … order_id = ”.$orderId;

となる。これを実行してはまずい。

↓

世間でいう「SQLインジェクション」はこういうPHPプログラムで起こっている!

DBと連携するシステムをプログラムするとき、プログラム内のSQL文は文字列結合で作成してはダメ!
バインド変数が可能なプリペアドステートメントとプレースホルダを使う!

使い方は以下の手順。

1. SQL文で変数にあたる部分を適当な文字列を使って「:○○」(プレースホルダ)に置き換える。
   $sql = “DELETE … order_id = :order_id”;
2. 変数のバインドステップでこのプレースホルダに値を埋め込むメソッドを実行する。
   $stmt->bindValue(“:order_id”, $orderId, PDO::…);
   ↑ ↑ ↑
   プレースホルダ名 埋め込む値(変数) (下記参照)

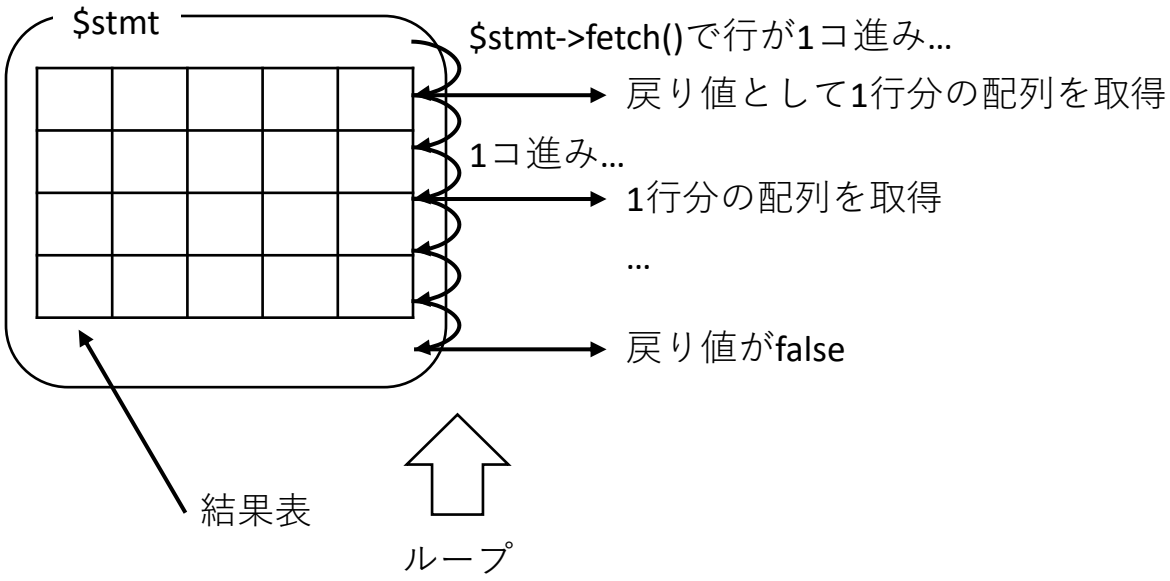
bindValue()の第3引数は、カラムのデータ型に応じて以下の定数を指定する。

- PDO::PARAM_INT: 数値型
- PDO::PARAM_BOOL: ブール型(true/false)
- PDO::PARAM_STR: その他
- PDO::PARAM_NULL: null値

(注)
SQL文上で‘’(シングルクオート)で囲んでいる変数(文字列など)の場合、
プレースホルダに置き換えるときにシングルクオートを書かない。
変数のバインド時に自動でシングルクオートが付与される。
(ex.)
SELECT … order_mode = ‘online’;
↓
$sql = “SELECT … order_mode = :order_mode”;

(3) データ取得

PDOStatementオブジェクトでSELECT文を実行した場合、内部にその結果表が丸々格納されている。



上記ループ処理をコードにすると、次のようになる。

```
$row = $stmt->fetch();   ←1行分のデータが格納された配列を取得
while($row) {   ←$rowがfalseになるまで
   $rowを使ってデータ取得
   $row = $stmt->fetch();   ←次の行を取得
}
```

このうち、$rowの取得をwhileの条件式にまとめることができるから…

データ取得の手順は以下の通り。

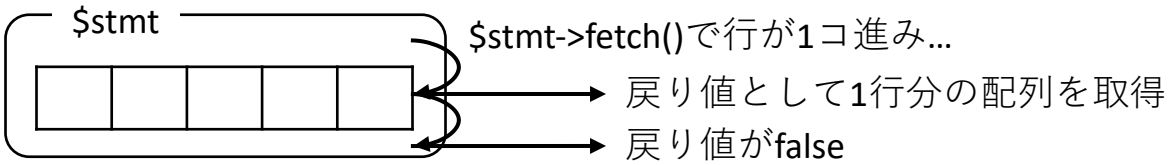
1. whileループで1行ずつ取り出す。
```
while($row = $stmt->fetch()) {
    :
}
```
2. ループ内で、1行分の配列($row)から各カラムのデータを取り出す。キーとしてカラム名を指定する。
```
$orderId = $row[“order_id”];
                    ↑
                カラム名
```

(注)
fetch()メソッドは以下の3コの引数をとることができる。詳しくはWebで。
1. $mode: フェッチモード(定数値を使用)
2. $cursorOrientation: カーソルの移動方向(定数値を使用)
3. $cursorOffset: フェッチを開始する行番号

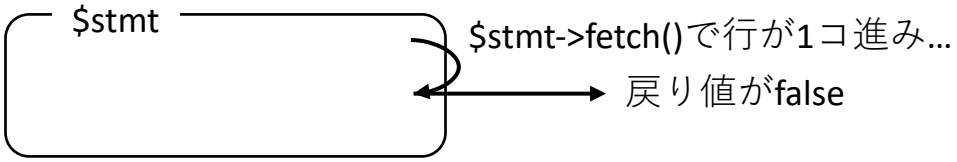
[5] データ取得のTIPS

(1) 結果が1行の場合

SELECT文の結果が明らかに1行とわかっている場合…



か



のどちらか

なので…

SELECTの結果が明らかに1行の場合、ループを使わず、以下の書き方が可能。

```
if($row = $stmt->fetch()) {
        データ取得処理
}
else {
        結果がない場合の処理
}
```

(2) LIKE検索のコツ

LIKEを使って検索を行う場合…

SELECT … WHERE cust_first_name LIKE '%ar%';

をプレースホルダに置き換えるとき、「ar」が変数だから、として

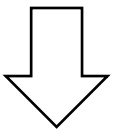
$sql = "SELECT … LIKE '%:keyword%';

とするとダメ!

変数をバインドするときに、PDOStatementが自動的にシングルクオートを付与するので、

SELECT … LIKE '%'ar'%'

というSQLができて、結果がないかエラーとなってしまう…orz。



なので…

LIKE検索SQLは%を含めてプレースホルダへの置き換える!

SELECT … WHERE cust_first_name LIKE '%ar%';

↓

$sql = "SELECT … WHERE cust_first_name LIKE :cust_first_name";

としておいて、

$stmt->bindValue(":cust_first_name", "%".$keyword."%", …);

とバインド時に%を付与する。