第８回　データベース接続（SELECT）

８－１．データベース接続

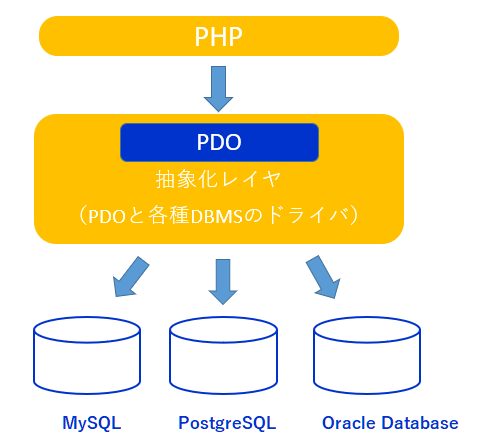
８－１－１．PDOとは

PHPからSQLを実行するために、PDO（PHP Data Object）を使用します。

PDOとは、PHPの拡張モジュールです。データベース製品の違いを吸収するための統一されたインターフェース（PDOクラス）を提供します。

　従来は様々なデータベース（postgreSQL, Microsoft SQL Server, Oracle Database）などにより、プログラムの書き方が異なりました。そのため、利用するデータベースを変更すると、PHPプログラムも大幅に修正する必要がありました。

　このような手間を省くため、PHPとデータベース（正確にはDBMS）の間に抽象化レイヤを挟んで、各種DBMSの違いを抽象化レイヤで吸収し、DBMSが異なっても同じ処理をできるようにするというのが、PDOの役割です。抽象化レイヤには、PDOとPDOが使用する各種ドライバが含まれています。



**＜＜オブジェクト指向言語としてのPHP＞＞**

今までの課題は、従来からの手続き型的な書き方だったが、PHP３（今使用しているのはPHP８）からオブジェクト指向への対応が始まり、現在ではオブジェクト指向的な開発ができるようになっている。

データベースアクセスについては、手続き型的に記述することも可能だが、現在は以下で説明するPDOでの開発が主流なので、それを使用していく。

PHPでのオブジェクト指向の書き方は、Javaと似ているので、特に詳細に説明はしないが、今回使用する部分だけ少し説明をする。

①　インスタンス化・・・・インスタンス化するには、new演算子を使用する。

$db = **new** PDO($dsn, DB\_USER, DB\_PASS);//PDOクラスのインスタンス化

②　インスタンスメソッドの実行・・・・・インスタンスメソッドの実行は　->　（アロー演算子）を使う

$db**->**setAttribute(PDO::ATTR\_EMULATE\_PREPARES, false);

③　クラスメソッドの実行・・・・・クラスメソッドの実行は　：：　（コロン2つ）を使う（今回は使用しない）

④　例外が発生する可能性がある場合は、try～catchで例外処理をする。

＜＜今回使用するテーブルについて＞＞

今回の課題（課題８～１０）で使用するテーブルは、データベース演習で使用しているstudb内のoldproduct表である。

（テーブル作成の準備：別紙参照）

テーブル構成

|  |  |  |  |
| --- | --- | --- | --- |
| カラム名 | データ型 | Null許可 | 制約等 |
| PRODUCT\_NO | char(4) | NO | P-Key　4桁数字 |
| PNAME | varchar(50) | NO | 入力必須 |
| CATEGORY | varchar(45) | NO | ピザかドリンクどちらか |
| PRICE | int | NO | ０～10000までの数値 |

赤字はプログラムでチェックする項目

８－１－２．データベースへの接続

データベース処理は下記のような手順になります。Sampleコードと併せて確認していきましょう。

1. PDOクラスをインスタンス化する
2. PDOの動作オプションを指定する
3. SQL文の準備と実行
4. SQL実行結果の処理
5. PDOオブジェクトを破棄
6. PDOインスタンス化（データベースへの接続）

■構文

new PDO([接続先DB情報], [DBのログインID], [DBのパスワード])

コンストラクタの引数は、３つです。

　接続先DB情報を表す文字列を、DSN（Data Source Name）といいます。

データベースにより書式が異なりますが、MySQLの場合は下記のようになります。

mysql:host=[接続先ホスト名またはIPアドレス]; dbname=[接続先DB名]; charset=[接続時の文字エンコーディング名]

＊緑色部分には環境に合わせた値を記載します。

＜＜今回の課題で使用するテーブル等＞＞

・ホスト名　　　　　：　localhost

・DB名　　　　　　：　studb

・DBユーザー　　：　dbuser

・DBパスワード　：　ecc

・テーブル 　：oldproduct

1. PDOの動作オプションを指定

$インスタンス変数->setAttribute([オプション種別], [オプション値])

＊今回設定するのは、PDO::ATTR\_ERRMODEとPDO::ATTR\_EMULATE\_PREPARESの2つ。

PDO::ATTR\_EMULATE\_PREPARESは、falseを設定する（静的プレースホルダー）

■setAttributeメソッドで指定できるオプション定数

|  |  |  |  |
| --- | --- | --- | --- |
| オプション種別 | 指定方法 | 意味 | デフォルト値 |
| PDO:ATTR\_CASE | 定数で指定 | SELECT結果のカラム名について大文字／小文字の扱いを指定 | PDO::CASE\_NATURAL |
| PDO::ATTR\_ERRMODE | 定数で指定 | エラー通知方法を指定 | PDO::ERRMODE\_SILENT |
| PDO::ATTR\_ORACLE\_NULLS | 定数で指定 | NULLと空文字の変換方法を指定。Oracle以外でも使用可能 | PDO::NULL\_NATURAL |
| PDO::ATTR\_AUTOCOMMIT | 真偽値で指定 | 自動コミットするか否かを指定 | true |
| PDO::ATTR\_EMULATE\_PREPARES | 真偽値で指定 | プリペアドステートメントのエミュレーションを有効にするか否かを指定 | true |
| PDO::ATTR\_DEFAULT\_FETCH\_MODE | 定数で指定 | SELECT結果のPHO変数へのマッピング方法を指定 | PDO::FETCH\_BOTH |

■PDO：：ATTR\_CASEに指定できる値

|  |  |
| --- | --- |
| 定数名 | 意味 |
| PDO:CASE\_LOWER | 小文字に変換する |
| PDO::CASE\_NATURAL | 変換しない |
| PDO::CASE\_UPPER | 大文字に変換する |

■PDO::ATTR\_ERRMODEに指定できる値

今回は、PDO::ERRMODE\_EXCEPTIONを設定するので、SQL例外エラーを送出するので、ｔｒｙ～catchでの処理が必要となる。

|  |  |
| --- | --- |
| 定数名 | 意味 |
| PDO::ERRMODE\_SILENT | エラーを出力しない。PDO::errCodeメソッドでエラーコード取得のみ可能 |
| PDO::ERRMODE\_WARNING | E\_WARNING定数レベルのエラーを出力 |
| PDO::ERRMODE\_EXCEPTION | 例外（おもにＰＤＯException）をスローする |

■PDO::ATTR\_ORACLE\_NULLSに指定できる値

|  |  |
| --- | --- |
| 定数名 | 意味 |
| PDO::NULL\_NATURAL | 変換しない |
| PDO::NULL\_EMPTY\_STRING | 空文字をNULLに変換する |
| PDO::NULL\_TO\_STRING | NULLを空文字に変換する |

■PDO::ATTR\_DEFAULT\_FETCH\_MODEに指定できる値

|  |  |  |
| --- | --- | --- |
| 定数名 | 意味 | 結果セットの変数の参照例 |
| PDO::FETCH\_ASSOC | カラム名をキーとした連想配列を返す | $result[‘column’] |
| PDO::FETCH\_NUM | 0で始まる配列を返す | $result[0] |
| PDO::FETCH\_BOTH | FETCH\_ASSOCとFETCH\_NUMをミックスした配列を返す | $result[0] / $result[‘column’] |
| PDO::FETCH\_OBJ | カラム名をプロパティに持つインスタンスを返す | $result->column |

1. SQL文の準備と実行

まず、SQLを作成する。(OLDPRODUCT表を全件表示)

$ sql = "SELECT \* FROM OLDPRODUCT";

■ SQLの準備

$ PDOステートメント変数 = $インスタンス変数->prepare([SQL])

■SQLの実行

$ PDOステートメント変数->execute( )

＊戻り値は、SQLを実行した結果。今回は、SELECT文なので、戻り値は結果配列となる。その際、setAttributeメソッドで設定されたPDO::ATTR\_DEFAULT\_FETCH\_MODEの形式の配列となる。今回は特に設定していないので、デフォルト値のPDO::FETCH\_BOTHが適用されているので、カラム名、要素番号どちらでもアクセス可能。

＜＜PDOStatement クラスとは＞＞

プリペアドステートメントを表します。ステートメント実行後は関連する結果セットを表します。

＜＜プリペアードステートメントとは＞＞

プログラム上で動的にSQL文を生成する必要があるとき、可変部分を変数のようにしたSQL文をあらかじめ作成しておき、値の挿入は処理系に行わせる方式。 実行効率を向上させ、SQLインジェクションを防止する効果がある

1. SQL実行結果の処理

SELECTの場合、実行結果をfetchメソッドまたはfetchAllメソッドにて取得します。

SQLを実行すると、結果セット（メモリ上に作成された仮想的なテーブル）が作成されます。fetchメソッドは結果セットから1レコードずつ、fetchAllメソッドは全レコード分を取り出してPHPの変数に割り当てることです。

■（PDOStatement::fetchメソッド）

$ PDOステートメント変数->fetch([フェッチモード定数])

　　　　　　　　　または

$ PDOステートメント変数->fetchAll([フェッチモード定数])

\*フェッチモード定数に指定できる値は、「■PDO::ATTR\_DEFAULT\_FETCH\_MODEに指定できる値

」と同じです。今回は、PDO::FETCH\_ASSOCを指定して、連想配列として取得します。

fetchメソッドの方がPHPのメモリ使用量を抑えられます。

＜＜例　fetchメソッドを使用してSQLの結果を取得する＞＞

while ($rows = $stmt->fetch(PDO::FETCH\_ASSOC)) {

$result[] = $rows;

}

1. PDOオブジェクトを破棄（データベースの切断）

DBの処理が終了したら、接続を閉じましょう。接続の順番は以下の通りです。

1. PDOステートメント変数を閉じるにはnullを代入します。
2. PDOインスタンス変数を閉じるにはnullを代入します。

＊明示的にこれを行わなかった場合は、スクリプトの終了時に自動的に 接続が閉じられます。

＊＊＊＊＊＊＊＊＊＊課題８＿１　第一段階（全件検索）を作成してみよう＊＊＊＊＊＊＊＊＊＊＊＊＊

８－2．SQLインジェクションとプレースホルダー

＜＜SQLインジェクションとは＞＞

データベースと連携したウェブアプリケーションの多くは、利用者からの入力情報を基にSQL文（データベースへの命令文）を組み立てています。ここで、SQL文の組み立て方法に問題がある場合、攻撃によってデータベースの不正利用をまねく可能性があります。このような問題を「SQLインジェクションの脆弱性」と呼び、問題を悪用した攻撃を、「SQLインジェクション攻撃」と呼びます。

（<https://www.ipa.go.jp/security/vuln/websecurity/sql.html>）

第2段階では、formから商品名（NAME）とカテゴリー（CATEGORY）を入力して検索ボタンを押すとその条件で検索結果が絞り込まれるようになっている。

つまりこの入力条件をもとに、WHERE句をSQL文に追加しなければならない。

その時に、ただ単純に入力内容を文字列連結したSQLを作成してしまうと、危険な文字列が入力された場合、最悪テーブルの内容が壊されてしまうことがある。

＜＜SQLインジェクションの対策＞＞

SQL文の組み立ては全てプレースホルダで実装する。

SQLには通常、プレースホルダを用いてSQL文を組み立てる仕組みがあります。SQL文の雛形の中に変数の場所を示す記号（プレースホルダ）を置いて、後に、そこに実際の値を機械的な処理で割り当てるものです。ウェブアプリケーションで直接、文字列連結処理によってSQL文を組み立てる方法に比べて、プレースホルダでは、機械的な処理でSQL文が組み立てられるので、SQLインジェクションの脆弱性を解消できます。

PHPでは、作成するSQLのユーザーが設定した値の入っている変数値を入れる箇所をプレースホルダとして設定しておきます。そのまま文字列に変数名を連結しないようにしてください。

例：＊赤字の箇所がプレースホルダです。冒頭にコロンを付けることでプレースホルダを表します。

$ PDOステートメント変数 = $インスタンス変数->prepare(‘SELECT \* FROM PRODUCT WHERE NAME = :name’);

prepareメソッドを実施した後に、プレースホルダーに実際の値をバインドする必要がある。

■構文　プレースホルダに値をバインドする

$PDOステートメント変数->bindParam([プレースホルダ名], [バインドする値], [値のデータ型])

■値のデータ型に指定できる定数

|  |  |
| --- | --- |
| 定数名 | 意味 |
| PDO::PARAM\_BOOL | 真偽値型 |
| PDO::PARAM\_NULL | NULL |
| PDO::PARAM\_INT | 整数型 |
| PDO::PARAM\_STR | 文字列型、または小数型 |
| PDO::PARAM\_LOB | ラージオブジェクト型 |

その後、executeメソッドを実行して、SQLを実施する。

＜＜参考　LIKE句を含むSQLのプレースホルダー＞＞

今回の課題では、商品名の検索が部分検索（LIKE検索）になっている。LIKEでは、%、\_を使用するが、

その場合、%\_を含む文字列をあらかじめ作成し、それをプレースホルダーとして設定する必要がある。

// "%"を含めたLIKE句で設定する文字列を作成

$nameLike = "%" . $name . "%";

// $sqlを設定

$ｓｑｌ = " select \* from OLDPRODUCT WHERE PNAME like :name";

// 値をバインド

$stmt->bindParam(':name',$nameLike, PDO::PARAM\_STR);

＊＊＊＊LIKE句に関して＊＊＊＊＊＊

LIKE句であいまい検索をする場合、SQLの中に%や\_を使用する。なので、何も対策をしないと、%などの文字を含むデータの検索ができない。そのため、検索文字列の中身%や\_を含めたい場合は、それらの文字をエスケープする必要がある。できる方は、自分で調べてやってみよう。

＜＜参考　WHERE句の条件式が可変長の時のプログラム＞＞

今回の課題は、入力項目が2つあり、商品名のみ、カテゴリのみ、両方、と3パターンの入力がある。

それによってWHERE句以下の文が変わる。AND（またはOR）が条件式の前につくかつかないかの違いではあるが、それぞれのパターンに合わせて文を作成するのは手間になる（検索項目がもっと増えればパターンも増えるのでもっと手間が増える）。

そこで、「WHERE 1=1」を指定することで、すべてANDを付けた文を追加していくだけになるので、処理が簡素になる（ちなみに、1=1はtrueになるので、この条件自体は特に意味はない）。

SELECT \* FROM テーブル名 WHERE 1=1;

SELECT \* FROM テーブル名 WHERE 1=1 AND 抽出条件;

SELECT \* FROM テーブル名 WHERE 1=1 AND 抽出条件 AND 抽出条件;