

Webアプリ動作環境

...

PHP(mod_php)

Apache + PHP(mod_php) + MySQL

サービス構成

PHPで動作するWebアプリのサービス構成は

- Webサーバー：Apache
- Webアプリ動作方法(アプリケーション・サーバー)：mod_php
- データベース：MySQL or SQLite

これらのサービスをDockerのコンテナで動作させます。データベースはMySQLもしくはSQLiteを使用します。

【注意点】

コンテナ間で接続する場合はIPアドレスでなく、サービス名かコンテナ名で接続します。MySQLとPHPで接続を行う場合のホスト名は、localhostではなくサービス名かコンテナ名で接続します。

ファイル構成

環境構築のためのファイル構成は次のようになります。

```
docker-lamp          // 開発環境ディレクトリ
├── docker-compose.yml // Dockerイメージとコンテナを生成・起動するための情報
├── htdocs            // Webルートディレクトリ
│   └── info.php      // PHP動作確認用
└── php              // Dockerfile、php.iniの格納ディレクトリ
    ├── Dockerfile    // PHPとApacheの融合コンテナを生成するための情報
    └── php.ini        // PHP設定ファイル
```

docker-compose.yml ①

YAML形式のファイルを作成し「docker-compose」コマンドでファイルに記述されたコンテナのイメージ作成・起動を行います。

```
version: '3.8'
services:
  php-apache:
    build: ./php
    container_name: php-apache
    volumes:
      - ./htdocs:/var/www/html
    restart: always
    ports:
      - "8000:80"
    depends_on:
      - mysql
```

version:	Composeファイルのバージョン
services:	生成・起動するサービス(コンテナ)を設定
php-apache:	サービス名(任意の名前)
build:	Dockerfileのあるディレクトリを指定
container_name:	コンテナ名(任意の名前)
volumes:	コンテナ内とホストOSのディレクトリを関連付ける Webルートのディレクトリを指定
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは 停止するが、Dockerを起動すると自動で再起動する
ports:	公開用のポート番号(ホスト:コンテナ)
depends_on:	コンテナの起動順序（指定したサービスの後に起動）

docker-compose.yml ②

```
mysql:
  image: mysql:latest
  container_name: mysql
  volumes:
    - db_data:/var/lib/mysql
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: 'password'
  ports:
    - "3306:3306"
```

mysql:	サービス名(任意の名前)
image:	DockerHubにあるイメージをそのまま利用するとき指定
container_name:	コンテナ名(任意の名前)
volumes:	コンテナ内とvolumes:を関連付ける この設定でDBのデータを永続的に保存する
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは停止するが、Dockerを起動すると自動で再起動する
environment:	コンテナ内の環境変数を設定 MYSQL_ROOT_PASSWORD:はMySQLのrootパスワードを設定

docker-compose.yml ③

```
phpmyadmin:
  image: phpmyadmin:latest
  container_name: phpmyadmin
  depends_on:
    - mysql
  environment:
    PMA_HOST: mysql
    MEMORY_LIMIT: 128M
    UPLOAD_LIMIT: 64M
  restart: always
  ports:
    - "8080:80"
```

phpmyadmin:	サービス名(任意の名前)
image:	DockerHubにあるイメージをそのまま利用するとき指定
container_name:	コンテナ名(任意の名前)
depends_on:	コンテナの起動順序（指定したサービスの後に起動）
environment:	PMA_HOST:利用するDBのサービス(mysql) MEMORY_LIMIT:ツールを利用するときの最大メモリ容量 UPLOAD_LIMIT:ファイルをアップロードするときの最大サイズ
restart:	コンテナが停止すると常に再起動する(always) 例えばコンテナ動作中にDockerが終了するとコンテナは停止するが、Dockerを起動すると自動で再起動する
ports:	公開用のポート番号(ホスト:コンテナ) phpmyadmin利用時のポート番号

```
volumes:
  db_data: {}
```

volumes:	DBデータを永続化するための設定 db_dataの名前で保存される
----------	--------------------------------------

Dockerfile

php-apacheイメージを生成・起動するためのDockerfileを作成します。
php-apacheはmod_phpを含んだApacheを生成・起動します。

- ホストOSで作成したphp.ini(PHP設定)を有効にする
- 画像/日本語/MySQL接続の処理を行うためのライブラリをインストールする

```
FROM php:8.0-apache
COPY ./php.ini /usr/local/etc/php/
RUN apt-get update &&\
    #PHP GD Multibyte String
    apt-get install -y zlib1g-dev libpng-dev libwebp-dev libjpeg62-turbo-dev libonig-dev &&\
    docker-php-ext-configure gd --with-jpeg --with-webp &&\
    docker-php-ext-install -j$(nproc) gd &&\
    #PHP PDO MySQL
    docker-php-ext-install pdo_mysql mysqli
#mod_rewrite有効化
RUN a2enmod rewrite
```

PNG,JPEG,WEBP画像,日本語を扱うための
ライブラリをインストール

MySQL接続のためのライブラリをインストール

php.ini

PHPの設定ファイル

- 日本のタイムゾーン
- 日本語設定

[Date]

date.timezone = "Asia/Tokyo"

[mbstring]

mbstring.language = "Japanese"

サーバーの生成・起動

これで一通り必要な設定ファイルなどが完成したのでDocker Composeを使ってサーバーを生成・起動します。

Docker Desktopを起動し、次のコマンドをdocker-lamp配下で実行します。

【生成・起動】

```
docker-compose up -d
```

このコマンド実行でディレクトリにあるdocker-compose.ymlファイルに基づいてサービスが生成・起動されます。はじめての実行時にはイメージが生成されます。

【終了】

```
docker-compose down
```

このコマンドで起動しているサービスを停止します。この場合イメージは残り、DBは保存されます。

【終了（イメージとDBの削除）】

```
docker-compose down --rmi all --volumes
```

このコマンドでサービスを停止すると共に、イメージとDBも削除されます。

動作確認

サーバー環境の生成・起動ができた後、動作確認のため次のコードのphpファイルを作成しブラウザで「localhost:8000/info.php」にアクセスします。

info.php

```
<?php
phpinfo();
?>
```

※ロケールが日本になっていることを確認します。

[illegible]

ブラウザにPHPの
情報が表示され
ば動作している

Webアプリサンプル作成①

Dockerコンテナ間で接続する場合、ホスト名がサービス名orコンテナ名となります。

次のコードはWebアプリのサンプルです。<?php

(htdocs/sample.php)

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width,
initial-scale=1.0">
  <link href="css/style.css"
rel="stylesheet">
  <title>サンプル</title>
</head>
<body>
  <h1>サンプル</h1>
```

```
$now = new DateTime();
print $now->format('Y年m月d日G時i分s秒');

$dsn = 'mysql:host=mysql;dbname=sampled;charset=utf8';
$user = 'root';
$password = 'password';

try {
  $db = new PDO($dsn, $user, $password);
  print '<p>接続成功</p>';

  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $db->exec("CREATE TABLE IF NOT EXISTS users(
    id INTEGER PRIMARY KEY,
    name VARCHAR(20),
    score INTEGER)");
  print '<p>テーブル作成</p>';
```

Webアプリサンプル作成②

```
$db->exec("INSERT INTO users VALUES(1, 'Yamada', 85)");
$db->exec("INSERT INTO users VALUES(2, 'Tanaka', 79)");
$db->exec("INSERT INTO users VALUES(3, 'Suzuki', 63)");
print '<p>データ挿入</p>';

$q = $db->query("SELECT * FROM users WHERE score >= 70");
print '<p>70点以上選択</p>';
print "<p>";
while ($row = $q->fetch()) {
    print $row["id"] . " " . $row["name"] . " " . $row["score"] . "<br>";
}
print "</p>";
$db->exec("DROP TABLE users");
print '<p>テーブル削除</p>';

} catch (PDOException $e) {
    die ('エラー: ' . $e->getMessage());
}

?>
</body>
</html>
```

このサンプルでは次の処理をします。

- テーブル作成
- テーブルにデータ挿入
- テーブルからデータを抽出
- テーブル削除

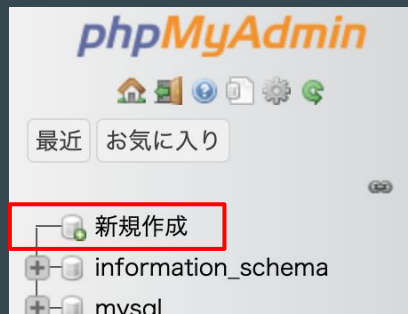
データベース作成

phpMyAdminを使ってデータベースを作成します。WebブラウザでURLを「localhost:8080」に接続するとログイン画面が表示されます。

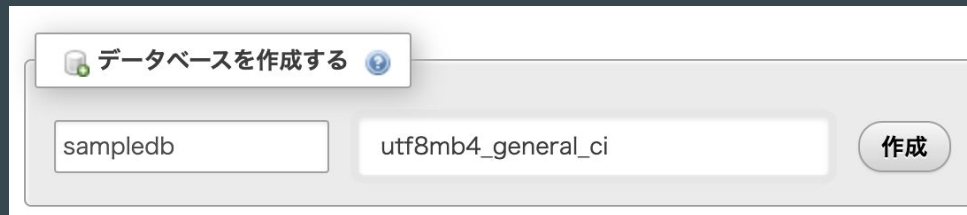


The login screen of phpMyAdmin. At the top is the phpMyAdmin logo and the text 'phpMyAdmin へようこそ'. Below this is a '言語 (Language)' section with a dropdown menu set to '日本語 - Japanese'. Underneath is a 'ログイン' section with input fields for 'ユーザ名:' and 'パスワード:', and a 'ログイン' button at the bottom right.

ユーザ名：root
パスワード：password



データベース名：sampedb
照合順序
：utf8mb4_general_ciで作成



The screen for creating a new database. It has a title bar that says 'データベースを作成する'. Below this are two input fields: the first contains 'sampedb' and the second contains 'utf8mb4_general_ci'. To the right of these fields is a button labeled '作成' (Create).

style.css

スタイルシート(css/style.css)

```
h1 {  
  color: red;  
}
```

Webルートであるhtdocs配下で任意の場所に配置します。
HTMLファイルやJavaScriptファイルなどの配置もhtdocs
配下で任意の場所となります。

```
htdocs  
├── css  
│   └── style.css
```

動作確認

Webブラウザで「localhost:8000/sample.php」にアクセスしサンプルコードで作成したページが表示されることを確認します

PHPはコンテナを再起動せずともソースコード修正後は、ブラウザをリロードすると最新の状態で表示します。

※現在日時が日本の日時になっていることを確認しましょう。

サンプル

2022年12月30日19時13分16秒

接続成功

テーブル作成

データ挿入

70点以上選択

1 Yamada 85

2 Tanaka 79

テーブル削除

MySQLとSQLiteのどちらも同じように動作します。違いがわかるようにするには見出しなどを修正しましょう。

Apache + PHP(mod_php) + SQLite

PHPでSQLiteを使用するには、PDOオブジェクトを作るときにSQLiteでデータベースファイルを指定します。

```
$db = new PDO("sqlite:./db/sample.db");
```

これは、コードが動作するカレントディレクトリ配下のdbディレクトリにデータベースとなるsample.dbファイルを作成します。

SQLiteはファイルがなければ作成します。（ディレクトリは存在しないとエラーになる）SQL文はMySQLと基本的に同じものが使えるので、サンプルのここを修正すればSQLiteを使用するようになります。

プレースホルダー

SQL文を実行するコードを記述する際、通常はSQLインジェクション対策としてプレースホルダーを使用します。（htdocs/sample2.php）

```
<!DOCTYPE html>
<html lang="ja">
<head>
  --省略--
  <title>サンプル2</title>
</head>
<body>
  <h1>サンプル2</h1>
  <?php
    --省略--
    $dsn = 'mysql:host=mysql;dbname=sampladb;charset=utf8';
    $user = 'root';
    $password = 'password';
    $options = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_EMULATE_PREPARES => false];
    $data = [[1, 'Yamada', 85], [2, 'Tanaka', 79], [3, 'Suzuki', 63]];
```

sample.phpを修正

このサンプルではデータをDBに挿入するときと、取得する処理にだけプレースホルダーを使用しています。

オプションを追加するので配列にしてPDO生成時に設定します。
ATTR_EMULATE_PREPARESをfalseにすることで静的プレースホルダーを使用するようになります。

プレースホルダー

```
try {
    $db = new PDO($dsn, $user, $password, $options); // MySQL
//    $db = new PDO("sqlite:./db/sample.db", options:$options); // SQLite
    print '<p>接続成功</p>';

    $db->exec("CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY,
        name VARCHAR(20),
        score INTEGER)");
    print '<p>テーブル作成</p>';

    $stmt = $db->prepare("INSERT INTO users (id, name, score) VALUES(:id, :name, :score)");
    foreach ($data as $value) {
        $stmt->bindParam(':id', $value[0], PDO::PARAM_INT);
        $stmt->bindParam(':name', $value[1], PDO::PARAM_STR);
        $stmt->bindParam(':score', $value[2], PDO::PARAM_INT);
        $stmt->execute();
    }
    print '<p>データ挿入</p>';
```

MySQLとSQLiteでPDO生成時の引数の数が異なります。MySQLは4つ目の引数にオプションを指定し、SQLiteはoptions引数にオプションを指定します。

bindParamは第2引数に変数で指定し、参照渡しとなるためリテラル値は使えません。値の評価はexecuteが実行されたときに行われます。

プレースホルダー

```
$stmt = $db->prepare("SELECT * FROM users WHERE score >= :score");
$stmt->bindValue(':score',70,PDO::PARAM_INT);
$stmt->execute();
print '<p>70点以上選択</p>';
print "<p>";
while ($row = $stmt->fetch()) {
    print $row["id"] . " " . $row["name"] . " " . $row["score"] . "<br>";
}
print "</p>";

$db->exec("DROP TABLE users");
print '<p>テーブル削除</p>';

} catch (PDOException $e) {
    die ('エラー: ' . $e->getMessage());
}
?>
</body>
</html>
```

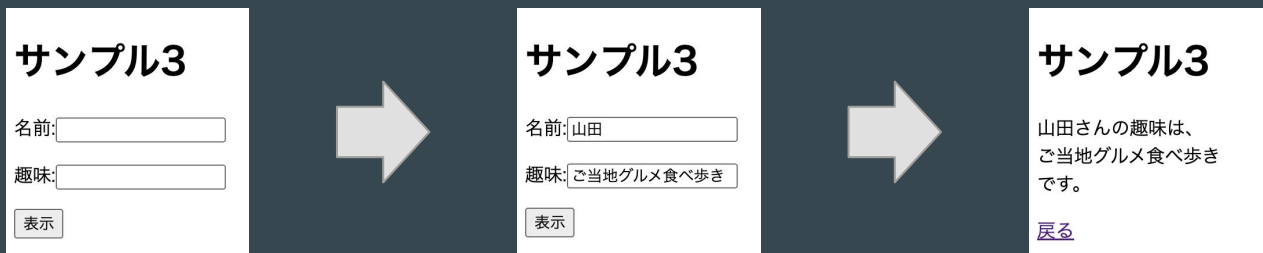
bindValueは第2引数に変数またはリテラル値で指定し、値の評価はこの関数が実行されたときに行われます。

HTMLエスケープ

フィールドに入力したデータを扱うときは注意が必要です。例えば、入力したデータを画面に表示する場合、データがスクリプトを実行するようなデータのときは意図しないプログラムが実行されるかもしれません。このような挙動を利用した攻撃をクロスサイト・スクリプティング(XSS)攻撃といいます。

対策として入力したデータをHTMLの特殊文字にエスケープすることで、HTMLのタグとしてではなく文字として扱われます。

ここではフィールドに入力したデータをエスケープするサンプルを作成します。



HTMLエスケープ

sample3.php

```
<?php
$name = "";
$hobby = "";
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $name = htmlspecialchars($_POST['name']);
    $hobby = htmlspecialchars($_POST['hobby']);
    if ($name == "" || $hobby == "") {
        print '<p>空白のフィールドがあります。</p>';
    }
}
?>
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>サンプル3</title>
</head>
```

POST送信で受け取ったデータをhtmlspecialcharsを使ってHTMLエスケープします。

HTMLエスケープ

```
<body>
  <h1>サンプル3</h1>
  <?php if ($name != "" && $hobby != ""): ?>
    <p>
      <?= $name ?>さんの趣味は、<br>
      <?= $hobby ?><br>
      です。
    </p>
    <a href="">戻る</a>
  <?php else: ?>
    <form action="<?= htmlspecialchars($_SERVER['PHP_SELF']) ?>" method="POST">
      <p><label>名前:<input type="text" name="name"></label></p>
      <p><label>趣味:<input type="text" name="hobby"></label></p>
      <button type="submit">表示</button>
    </form>
  <?php endif; ?>
</body>
</html>
```

`$_SERVER['PHP_SELF']`を使用し自身のURLを取得するときは、`htmlspecialchars`でエスケープします。これはURL末尾にスクリプトタグが埋め込まれていた場合に実行してしまうのを防ぎます。

動作確認

サンプル3

名前:

趣味:

表示



サンプル3

名前:

趣味:

表示



サンプル3

山田さんの趣味は、
ご当地グルメ食べ歩き
です。

[戻る](#)

サンプル3

名前:

趣味:

表示



サンプル3

<script>alert('XSS');</script>さんの趣味は、
<script>alert('XSS');</script>
です。

[戻る](#)

スクリプトが実行できるタグを
入力しても文字として表示され
ることを確認

エラーの確認について

PHPはデフォルト設定では通知や警告について出力しません。通知や警告を含めてエラー出力するには、次のコードをソース先頭に記述します。

```
error_reporting(E_ALL);  
ini_set('display_errors', '1');
```

PDOに関するエラー情報詳細は次のコードでも確認することができます。

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