# **SNSシステム開発 - リファクタリング**

最後にLearnSNSの仕上げに、リファクタリングを進めていきます。

## **概要**

リファクタリングとは、プログラムの外部から見た動作を変えずにソースコードの内部構造を整理することです。リファクタリングには様々なアプローチがありますが、今回はコードを読みやすいように改修を加えていきましょう。

リファクタリングを題材に下記について学習していきます。

1. 関数
2. 関数の定義と呼び出し

関数を利用して、まとまった処理毎に切り出し、再利用可能にすることでソースコードは飛躍的にキレイに、そして読みやすくなります。

## **要件確認**

要件はシンプルです。

* 挙動を変えずにコードを変える

## **関数について**

最初に、関数について確認していきましょう。

ユーザーのIDを渡すと該当するユーザー情報が返ってくる、などが関数となっていきます。

もう少し分かりやすい例を出すと、自販機によく例えられます。お金を投入すると、飲み物が返ってくる、という流れが関数を表現しています。



つまり、必要な値を関数に渡すと、定まったパターンで処理を加え、何らかの期待された結果を返します。

**関数とは、与えられた値に対して定められた処理を行って結果を返す機能**のことです。

また、この時、与えられる値のことを**引数(ひきすう)**、返ってくる結果を**戻り値(または、返り値)**、と言います。

これらはプログラミングにおいて非常に重要かつ基礎となる知識なので必ず覚えましょう。

では、ここから実際にtimeline.phpの処理を関数化していきます。

## **関数用ファイル**

まずは関数用のファイルを確認しましょう。

これから関数を記述していくファイルはfunction.phpになります。既にファイルは存在しているので、timeline.phpから呼び出してみましょう。

**timeline.php**

<?php

require('dbconnect.php');

require('function.php');

この時点でタイムライン画面を開き、エラーが起きていないことを確認しましょう。

## **関数定義**

**ユーザー情報取得処理**

まずはサインインしているユーザーを取得する処理を関数にしてみましょう。

**timeline.php**

$sql = 'SELECT \* FROM `users` WHERE `id`=?';

$data = [$\_SESSION['LearnSNS']['id']];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

$signin\_user = $stmt->fetch(PDO::FETCH\_ASSOC);

上記の処理を関数にします。

関数の定義方法は、下記の通りです。

function 関数名(引数) {

処理

return 返り値;

}

早速、一つ関数を定義してみましょう。

**function.php**

<?php

function getUser($dbh, $id) {

$sql = 'SELECT \* FROM `users` WHERE `id`=?';

$data = [$id];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

return $stmt->fetch(PDO::FETCH\_ASSOC);

}

関数を定義出来たらtimeline.phpから利用してみましょう。

関数の利用方法は、下記の通りです。

関数名(引数);

関数の戻り値を利用する場合は下記になります。

$変数 = 関数名(引数);

自動販売機の例で考えると、$飲み物 = 自販機(お金);となるでしょう。

では、timeline.phpから定義した関数を利用してみましょう。

**timeline.php**

~~$sql = 'SELECT \* FROM `users` WHERE `id`=?';~~

~~$data = [$\_SESSION['LearnSNS']['id']];~~

~~$stmt = $dbh->prepare($sql);~~

~~$stmt->execute($data);~~

~~$signin\_user = $stmt->fetch(PDO::FETCH\_ASSOC);~~

↓

$signin\_user = getUser($dbh, $\_SESSION['LearnSNS']['id']);

これまで記述していた内容は関数に切り出したので不要です。削除しましょう。

実装が出来たらブラウザで挙動に変更がないことを確認しましょう。

なお、今回関数に置き換えた処理はサインインユーザーの情報を取得する処理ですが、指定したIDのユーザー情報を取得できれば対応できるので、今回はgetUserとして実装しました。

**投稿情報登録処理**

続いて、投稿情報登録処理を関数にしていきます。

**timeline.php**

$sql = 'INSERT INTO `feeds` (`feed`, `user\_id`, `created`) VALUES (?, ?, NOW())';

$data = [$feed, $signin\_user['id']];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

上記の処理を関数に切り出していきましょう。

**function.php**

function createFeed($dbh, $feed, $user\_id) {

$sql = 'INSERT INTO `feeds` (`feed`, `user\_id`, `created`) VALUES (?, ?, NOW())';

$data = [$feed, $signin\_user['id']];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

}

なお、今回記述するfunctionは前回書いたgetUserとは全く違う関数になるので、入れ子にならないように気をつけましょう。

また、今回はreturn文がありません。投稿情報を登録した後の情報は必要がないため戻り値もありません。戻り値が必要ない場合、**return文を省略することが出来ます**。

関数が書けたらtimeline.phpもあわせて修正しましょう。

**timeline.php**

~~$sql = 'INSERT INTO `feeds` (`feed`, `user\_id`, `created`) VALUES (?, ?, NOW())';~~

~~$data = [$feed, $signin\_user['id']];~~

~~$stmt = $dbh->prepare($sql);~~

~~$stmt->execute($data);~~

↓

createFeed($dbh, $feed, $signin\_user['id']);

前回と同様、記述が出来たらブラウザで挙動に変わりがないことを確認しましょう。

ここで一つ注意しておきたいのが、引数の順番です。

関数の引数の指定は、定義した時と呼び出す時に同じ順番である必要があります。ただし、変数名は同じである必要はありません。

このように、各処理を関数にすることでソースコードの可読性(読みやすさ)が飛躍的に上がります。

以降は、手順が同じなので、ソースコードのみ掲載しておきます。

**いいね件数**

**function.php**

function countLike($dbh, $feed\_id) {

$sql = "SELECT COUNT(\*) AS `like\_cnt` FROM `likes` WHERE `feed\_id` = ?";

$data = [$feed\_id];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

$result = $like\_stmt->fetch(PDO::FETCH\_ASSOC);

return $result['like\_cnt'];

}

**timeline.php**

~~$like\_sql = "SELECT COUNT(\*) AS `like\_cnt` FROM `likes` WHERE `feed\_id` = ?";~~

~~$like\_data = [$record["id"]];~~

~~$like\_stmt = $dbh->prepare($like\_sql);~~

~~$like\_stmt->execute($like\_data);~~

~~$like = $like\_stmt->fetch(PDO::FETCH\_ASSOC);~~

~~$record["like\_cnt"] = $like["like\_cnt"];~~

↓

$record['like\_cnt'] = countLike($dbh, $feed\_id);

**いいね済みの判定**

**function.php**

function isLiked($dbh, $user\_id, $feed\_id) {

$sql = "SELECT `id` FROM `likes` WHERE `user\_id` = ? AND `feed\_id` = ?";

$data = [$user\_id, $feed\_id];

$stmt = $dbh->prepare($sql);

$stmt->execute($data);

$is\_liked = $stmt->fetch(PDO::FETCH\_ASSOC);

return $is\_liked ? true : false;

}

**timeline.php**

~~$like\_flg\_sql = "SELECT `id` FROM `likes` WHERE `user\_id` = ? AND `feed\_id` = ?";~~

~~$like\_flg\_data = [$signin\_user['id'], $record["id"]];~~

~~$like\_flg\_stmt = $dbh->prepare($like\_flg\_sql);~~

~~$like\_flg\_stmt->execute($like\_flg\_data);~~

~~$is\_liked = $like\_flg\_stmt->fetch(PDO::FETCH\_ASSOC);~~

~~$record["is\_liked"] = $is\_liked ? true : false;~~

↓

$record['is\_liked'] = is\_liked($dbh, $signin\_user['id'], $record['id']);

## **まとめ**

どうでしょうか。だいぶtimeline.phpがスッキリしたのではないでしょうか？

当然ながら関数に切り出すことができる処理は他にもたくさんあります。他のページから同じ関数を使うことも出来ます。

余裕のある方は是非チャレンジしてみてください！

また、リファクタリングには他にも様々な方法があります。

制御文のネストを浅くすることや条件式の簡略化、適切な変数名にリネーム、などなど。処理の順番を変えるだけでもコードが読みやすくなることもあります。

ソースコードをより良くするために、自身の書いたコードを疑う姿勢は、できるエンジニアの必須条件です。動いたら終わりではないのです。

## **お疲れ様でした！**

以上で、リファクタリングについて、そしてLearnSNSのカリキュラムが完成です。

これまで見てきたように、LearnSNSにはWeb基礎の全てが詰まっています。

最初に述べたように、このカリキュラムを何度も繰り返し繰り返し取り組んで、是非みなさんのモノにしていただきたいです。

もちろん、ゼロから一人でWebアプリを作ることは何よりも勉強になります。

作りたいものがある方は是非チャレンジしてください！

1周目の方

* 2周目に取り組み、忘れている内容がないか今一度確認しましょう。

2~3周目の方

* 解説を見ずに実装できるでしょうか？弱点を把握してしっかりものにしましょう。

4周目以降の方

* HTML&CSSから実装してみましょう。
* また、カリキュラムの内容を解説できるでしょうか？