DB update & delete

Contents

- webの仕組み（復習）
- DB接続の関数化
- PHPからDB操作
  - 更新
  - 削除
- 課題発表 -> P2Pタイム

rules...

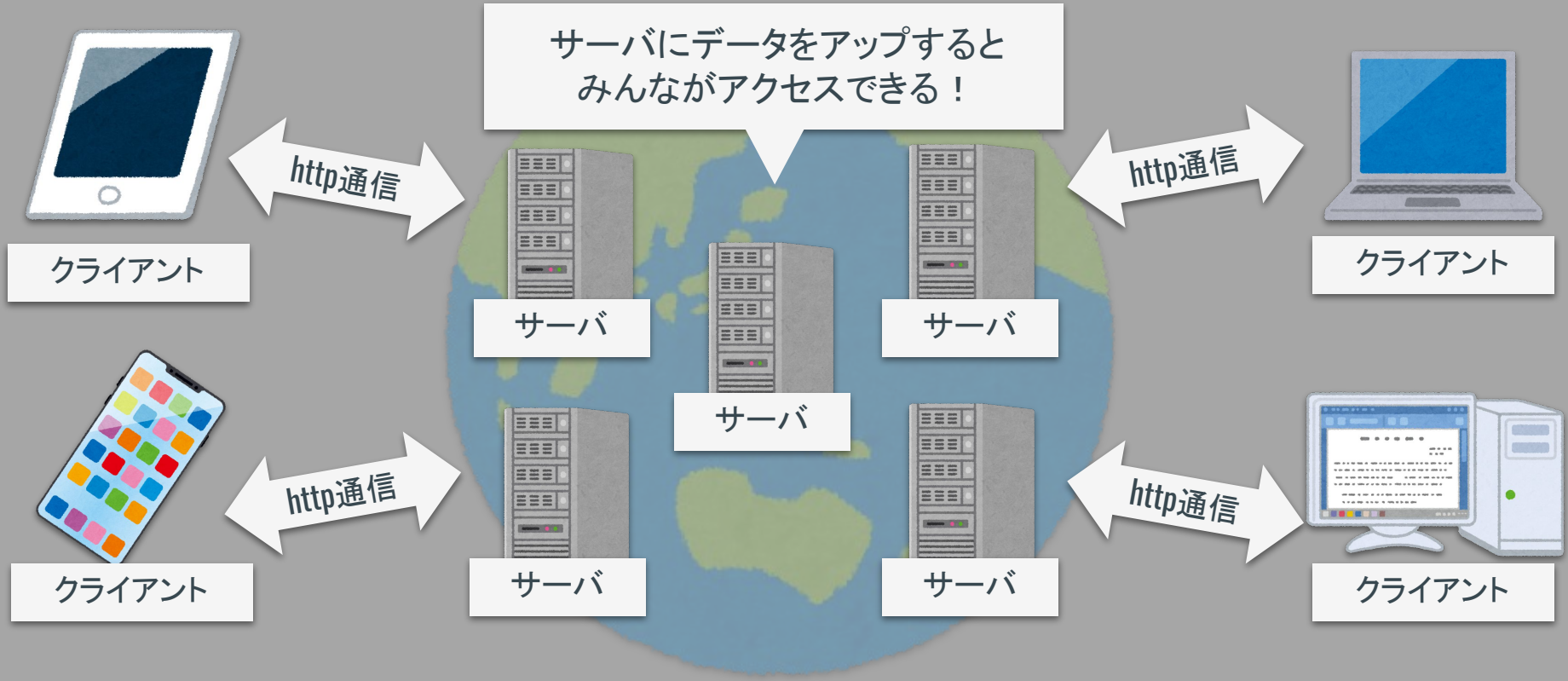
- 授業中は常にエディタを起動！
- 考えたことや感じたことはslackのガヤチャンネルでガンガン発信！
- 質問はslackへ！ 他の人の質問にも目を通そう！（同じ質問があるかも）
- 演習時，できた人はスクショなどslackに貼ってアウトプット！
- まずは打ち間違いを疑おう！
  - {} ' " ; など
- 書いたら保存しよう！（よく忘れる！）
  - command + s
  - ctrl + s

Goal

- CRUD処理の完成！
- 共通のコードを利用する！
- webサービスの構想を練る！

webの仕組み

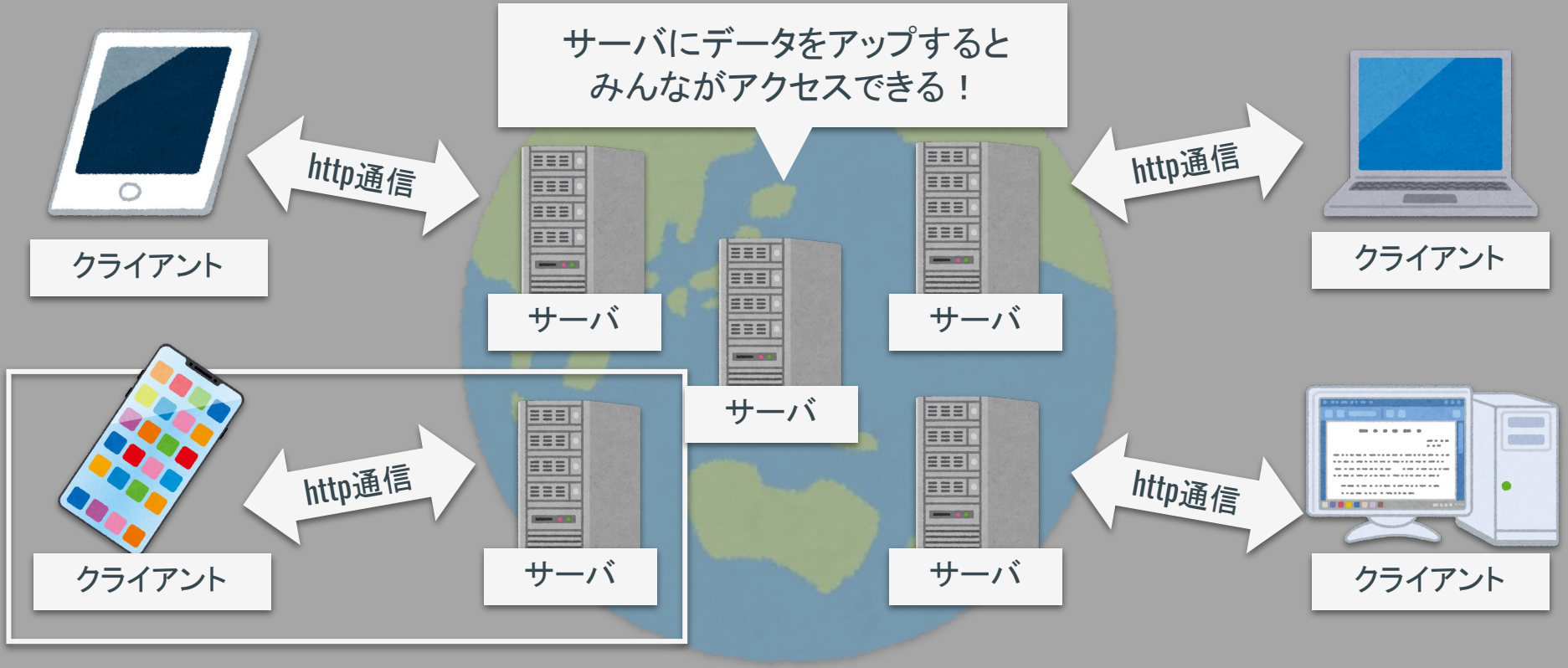
雑なwebの仕組み


サーバにデータをアップすると
みんながアクセスできる！
http通信
http通信
http通信
http通信
クライアント
クライアント
クライアント
クライアント
サーバ
サーバ
サーバ
サーバ
サーバ

URL

- URLとは
  - web上にある情報（ファイル）の場所を指し示す住所.
  - Uniform Resource Locatorの略（覚えなくてOK）.
- 例

雑なwebの仕組み


サーバにデータをアップすると
みんながアクセスできる！
http通信
クライアント
サーバ
サーバ
サーバ
http通信
クライアント
http通信
クライアント
サーバ
サーバ
http通信
クライアント

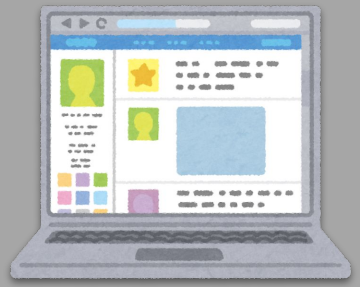
サーバとクライアント

- サーバで動作する言語（サーバサイド）
  - サーバ上でプログラムが実行される.
  - PHP, ruby, python, JAVA, (node.js), etc...
- クライアント（webブラウザ）で動作する言語（クライアントサイド）
  - webブラウザがプログラムを実行する.
  - html, css, javascript

サーバ - クライアント型のアプリケーション

サーバサイド言語の動き方

※ 言語によらず，ファイル（プログラム）はサーバ上に存在



送られてきたhtmlを実行

- こういう情報がほしい
- こういう処理をしたい
  例：index.phpにアクセス

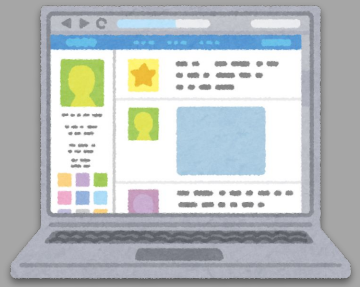
http通信

http通信

- 処理した結果のデータ
- 構成したhtml

クライアントサイド言語の動き方

※ 言語によらず，ファイル（プログラム）はサーバ上に存在



送られてきたhtmlを実行

- こういう情報がほしい
- こういう処理をしたい
  例：index.htmlにアクセス

http通信

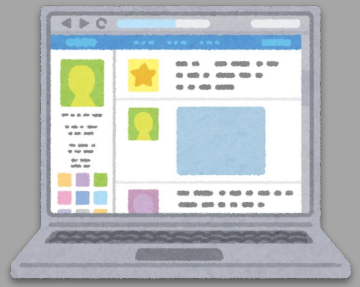
http通信

- リクエストしたhtml
- データを返すだけ！！

JavaScriptもこのタイミングで実行！！

DBの動き方

サーバ上のプログラム（PHPなど）がDBにアクセスして処理を実行！



送られてきたhtmlを実行

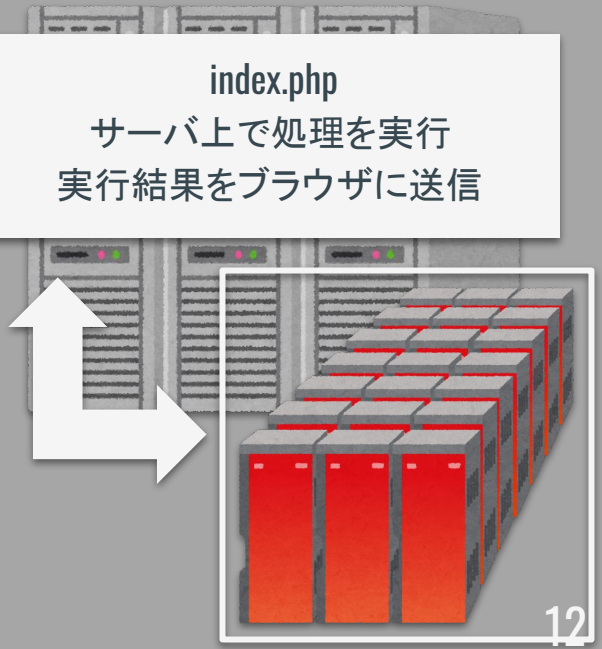
- こういう情報がほしい
- こういう処理をしたい
  例：index.phpにアクセス

http通信

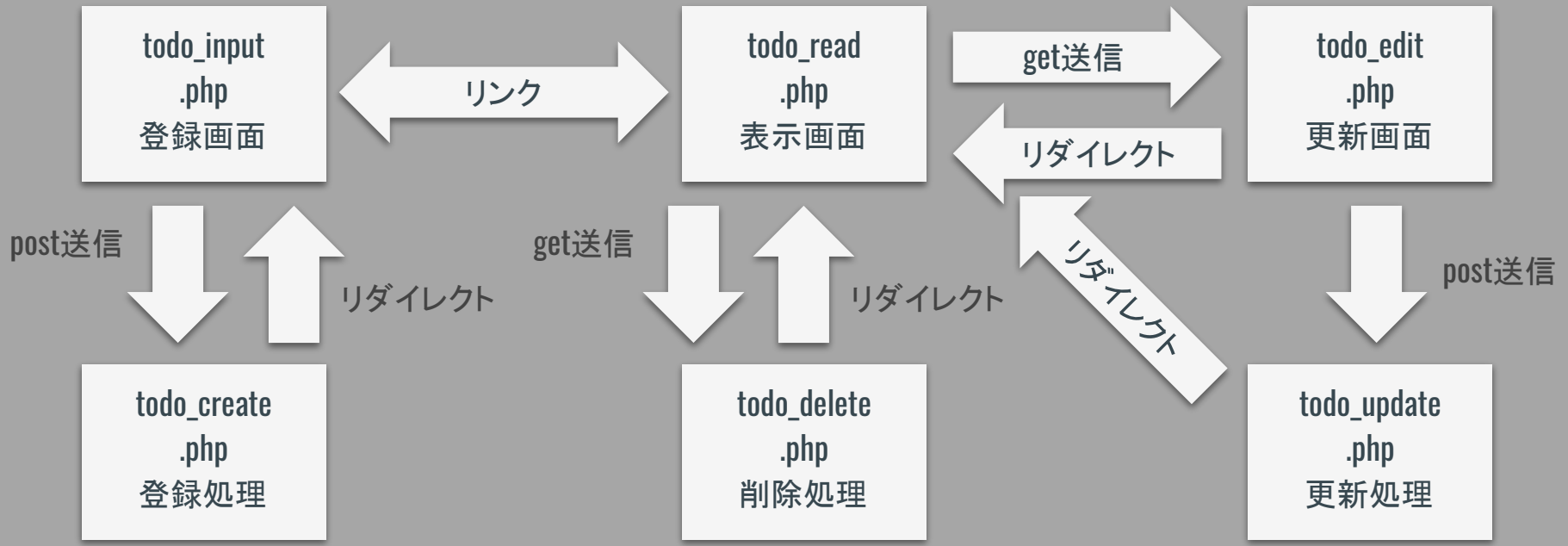
http通信

- 処理した結果のデータ
- 構成したhtml

index.php
サーバ上で処理を実行
実行結果をブラウザに送信



todoアプリの全体像



データ更新の前に. .

DB接続は常に同じコード．．！

実は．．．

- todo_input.phpとtodo_read.phpで記述したDB接続のコードは全く同じ！
- 今回作成するtodo_edit.php, todo_update.php, todo_delete.phpも同じ記述！

であれば．．！

- 一つの関数にまとめられる！
- 関数用のファイルを作成しよう！（functions.php）

DBに接続する関数を定義！（functions.php）

```
// 関数の定義
function connect_to_db(){      // DB名は自分で設定したものを使用する！
  $dbn='mysql:dbname=YOUR_DB_NAME;charset=utf8;
          port=3306;host=localhost';
  $user = 'root';
  $pwd = '';
  try {
      return new PDO($dbn, $user, $pwd);
  } catch (PDOException $e) {
      exit('dbError:'.$e->getMessage());
  }
}
```

DBに接続する関数を定義！（functions.php）

```
// 呼び出し（todo_create.php, todo_read.php, など）

include('functions.php');          // 関数を記述したファイルの読み込み
$pdo = connect_to_db();            // 関数実行

// 他のDB接続が必要なファイルでも上記の2行でOK！
```

DB接続関数を定義して使おう！

- functions.phpにDB接続の関数を定義しよう！
- todo_create.phpとtodo_read.phpでfunctions.phpをinclude()し，関数を実行しよう！

今まで通りの動きが確認できればOK！

（これまでやっていた処理を関数にしただけなので，やっていることは一緒）

PHPからDB操作

データ更新処理
（Update）

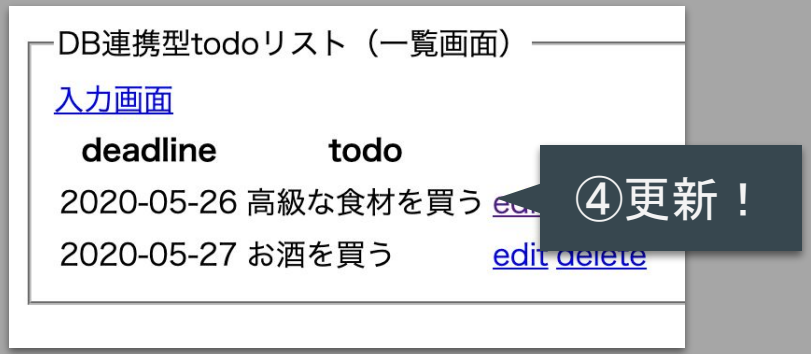
SQL構文：UPDATE（データ更新）

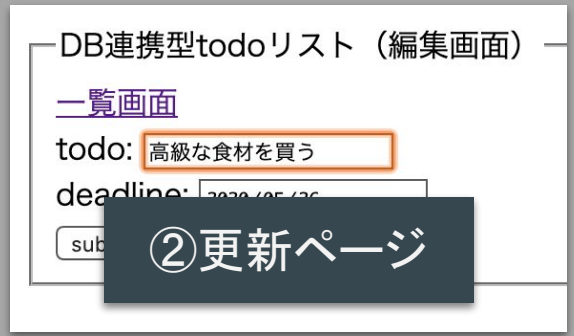
```
-- UPDATE文の基本構造
UPDATE テーブル名 SET 変更データ WHERE 選択データ;

-- 例
UPDATE gs_table SET name='gs99' WHERE id = 1;
--  【重要】必ずWHEREを使用！！（忘れると全てのデータが更新されます．．！）
```

データ更新処理

更新処理の流れ（登録処理と似ています！）

- ① 一覧画面に更新ページへのリンクを作成
  - urlにidを追加：todo_edit.php?id=**
- ② 更新ページの作成（todo_edit.php）
- ③ 更新処理の作成（todo_update.php）
- ④ 一覧画面に戻る





①一覧画面に更新ページへのリンクを追加（read.php）

```
foreach ($result as $record) {
  $output .= "<tr>";
  $output .= "<td>{$record["deadline"]}</td>";
  $output .= "<td>{$record["todo"]}</td>";
  // edit deleteリンクを追加
  $output .= "<td>
            <a href='todo_edit.php?id={$record["id"]}'>edit</a>
            </td>";
  $output .= "<td>
            <a href='todo_delete.php?id={$record["id"]}'>delete</a>
            </td>";
  $output .= "</tr>";
}
```

①動作確認（一覧画面）


DB連携型todoリスト（一覧画面）
入力画面
deadline todo
2020-05-26 食材を買う edit delete
2020-05-27 お酒を買う edit delete
editボタンにカーソルを載せて
「?id=**」になればOK！
localhost/php_tutorial/php03/php03_comp/todo_edit.php?id=1

②更新ページを作成（todo_edit.php）

```
// 関数ファイル読み込み
include("functions.php");

// 送信されたidをgetで受け取る
$id = $_GET['id'];

// DB接続&id名でテーブルから検索
$pdo = connect_to_db();
$sql = 'SELECT * FROM todo_table WHERE id=:id';
$stmt = $pdo->prepare($sql);
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$status = $stmt->execute();
```

該当のデータを取得するため
idでDBを検索する！

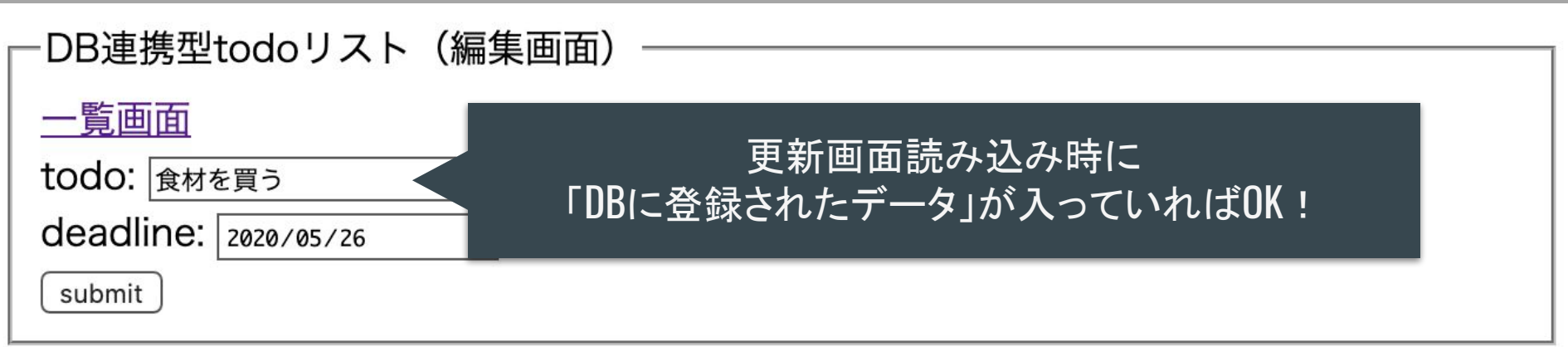
②更新ページを作成（todo_edit.php）

```
// fetch()で1レコード取得できる.
if ($status == false) {
  $error = $stmt->errorInfo();
  echo json_encode(["error_msg" => "{$error[2]}"]);
  exit();
} else {
  $record = $stmt->fetch(PDO::FETCH_ASSOC);
}
...
// htmlのタグに初期値として設定
<input type="text" name="todo" value="<?= $record["todo"] ?>">
<input type="date" name="deadline" value="<?= $record["deadline"]
?>">
```

②更新ページを作成（todo_edit.php）

```
// idを見えないように送る
// input type="hidden"を使用する！

// form内に以下を追加
<input type="hidden" name="id" value="<?=$record['id']?>">

// 更新のformは，登録と同じくpostで各値を送信しています！
```

②動作確認（編集画面）


DB連携型todoリスト（編集画面）
一覧画面
todo:
食材を買う
deadline:
2020/05/26
submit
更新画面読み込み時に
「DBに登録されたデータ」が入っていればOK！

③更新処理を作成 ④リダイレクト（todo_update.php）

```
// 各値をpostで受け取る
$id = $_POST['id'];
...

// idを指定して更新するSQLを作成（UPDATE文）
$sql = "UPDATE todo_table SET todo=:todo, deadline=:deadline,
            updated_at=sysdate() WHERE id=:id";
$stmt = $pdo->prepare($sql);
$stmt->bindValue(':todo', $todo, PDO::PARAM_STR);
$stmt->bindValue(':deadline', $deadline, PDO::PARAM_STR);
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$status = $stmt->execute();
```

③更新処理を作成 ④リダイレクト（todo_update.php）

```
// 各値をpostで受け取る

if ($status == false) {
  // SQL実行に失敗した場合はここでエラーを出力し，以降の処理を中止する
  $error = $stmt->errorInfo();
  echo json_encode(["error_msg" => "{$error[2]}"]);
  exit();
} else {
  // 正常に実行された場合は一覧ページファイルに移動し，処理を実行する
  header("Location:todo_read.php");
  exit();
}
```

更新処理を実装しよう

以下の処理を実装！

- 一覧画面にtodo_edit.phpへのリンクを追加！（todo_delete.phpも一緒に！）
- todo_edit.phpではデータをIDで検索し，該当するデータを表示！
- todo_update.phpでは入力したデータでUPDATE！
- 完了したらtodo_read.phpへ戻る.

更新処理実行後，一覧ページでデータが更新されていればOK！

（phpmyadminでも確認しよう）

データ削除処理
（Delete）

SQL構文：DELETE（データ削除）

```
// DELETE文の基本構造
DELETE FROM テーブル名;

// 例
DELETE FROM gs_table;                          -- 全消去
DELETE FROM gs_table WHERE id = 2;             -- 指定データのみ

// WHEREで指定しないとテーブルのデータが全滅する！！
// DELETEすると復旧できないので注意！！
```

削除処理

削除処理の流れ

- 済① 一覧画面に削除ページへのリンクを作成
  - urlにidを追加todo_delete.php?id=**
- ② 削除処理の作成（todo_delete.php）
- ③ 一覧画面に戻る

②削除処理を作成 ③リダイレクト（todo_delete.php）

```
// idをgetで受け取る
$id = $_GET['id'];

// idを指定して更新するSQLを作成 -> 実行の処理
$sql = 'DELETE FROM todo_table WHERE id=:id';
...

// 一覧画面にリダイレクト
header('Location:todo_read.php');
```

動作確認（削除処理）


DB連携型todoリスト（一覧画面）
入力画面
deadline
todo
2020-05-26 食材を買う edit delete
deleteクリックでデータが消えればOK！

更新処理を実装しよう

以下の処理を実装！

- 【済】一覧画面にtodo_delete.phpへのリンクを追加！
- todo_delete.phpではデータをIDで検索し，該当するデータを削除！
- 完了したらtodo_read.phpへ戻る.

削除処理実行後，一覧ページでデータが削除されていればOK！

（phpmyadminでも確認しよう）

課題

【課題1】卒業制作プロトタイプ（継続）

今回は「データ更新」「データ削除」の処理！

下記のポイントなどを自分のアイデアに合わせて考えてみよう！

- 「id」を指定してデータを特定する！
- 「論理削除」「物理削除」の使い分け
- 「更新処理」は本当に必要なのか．．？？

そろそろwebアプリケーションの全体が実装できるぞッ！

※そろそろ形にしておかないとキツイ．．！

【課題2】ユーザ管理機能の作成

ユーザ管理テーブル（<- 必ず作成，DBはこれまでのものを使用）

- テーブル名： users_table
- カラム名など

	#	名前	データ型	照合順序	属性	NULL	デフォルト値	コメント	その他	操作
☐	1	id 🔑	int(12)			いいえ	なし		AUTO_INCREMENT	変更 削除 その他
☐	2	username	varchar(128)	utf8mb4_unicode_ci		いいえ	なし			変更 削除 その他
☐	3	password	varchar(128)	utf8mb4_unicode_ci		いいえ	なし			変更 削除 その他
☐	4	is_admin	int(1)			いいえ	なし			変更 削除 その他
☐	5	is_deleted	int(1)			いいえ	なし			変更 削除 その他
☐	6	created_at	datetime			いいえ	なし			変更 削除 その他
☐	7	updated_at	datetime			いいえ	なし			変更 削除 その他

【課題2】ユーザ管理機能の作成

前スライドでつくったユーザのデータを管理する処理を実装！

- ユーザ作成処理
- ユーザー覧参照&表示処理
- ユーザデータ更新処理
- ユーザデータ削除処理
- （サービス管理者がユーザのデータを操作するイメージ）

課題1と課題2はそれぞれ独立でOK！

締切は厳守！！

P2Pタイム

まずはチーム内で解決を目指す！

訊かれた人は苦し紛れでも応える！！