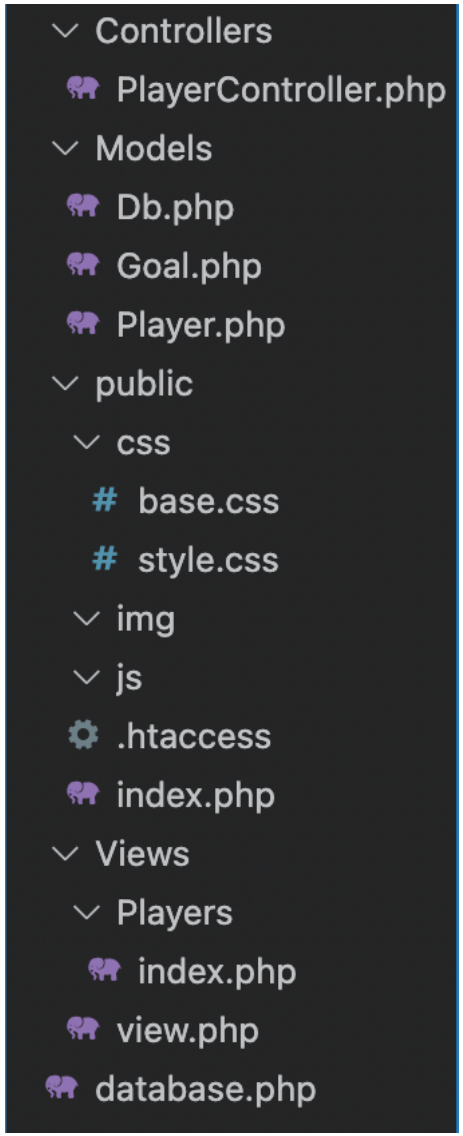
MVC 作成

はじめに

この課題では実際にクラス作成や継承を行いながらオブジェクト指向の活用イメージを広げていくことを目的としています。
前提としてオブジェクト指向の基礎知識がない状態だと、この課題に取り組んでもただのコピペになってしまい活用イメージに繋がらない可能性があります。
MVC 作成手順を進める前に必ず自分なりにオブジェクト指向の理解に努めてから行うよう にしてください。

MVC 作成

ディレクトリ構成



今回作るディレクトリ & ファイルです。
先ずは模倣しながら理解していきま しょう。(ファイル中身は空)

- Controllers/・・・処理振り分けを担当
- Models/・・・データ操作や DB 接続を担当
- Views/・・・見た目 (html 等) を担当
- public/・・・最初に読み込まれるルート
- .htaccess・・・Apache の設定ファイル
- database.php・・・DB 接続情報を記載

MVC 作成

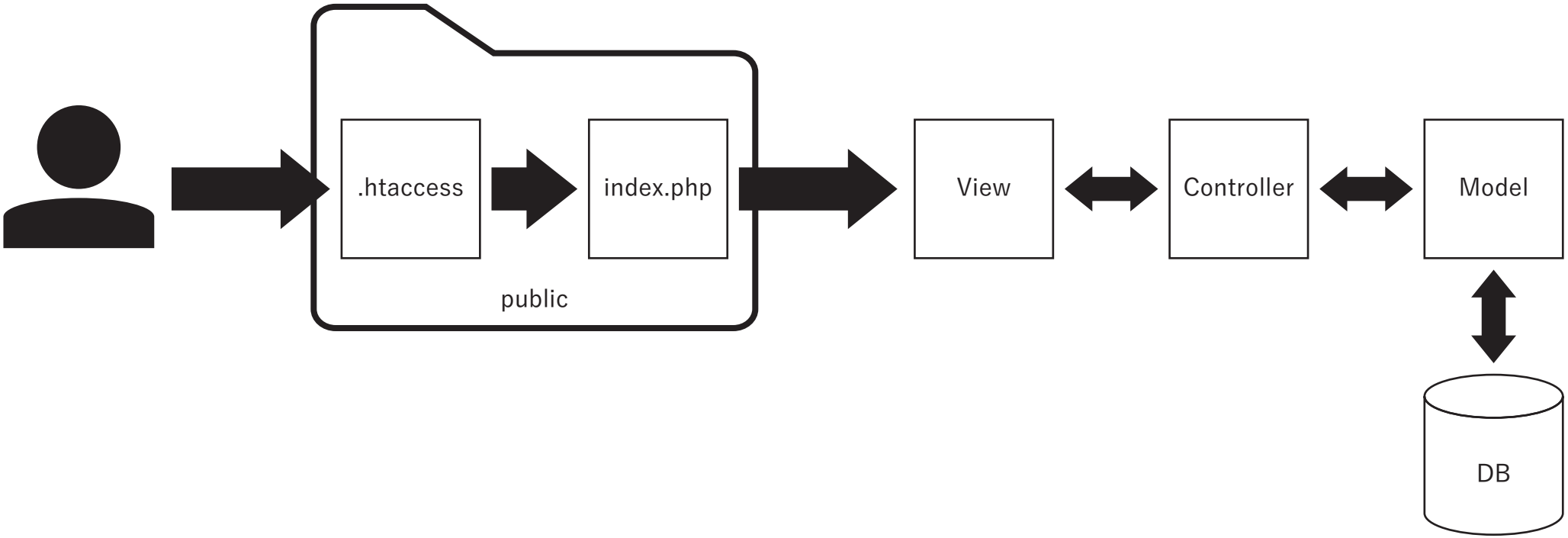
ドキュメントルート

MAMP or XAMPP のドキュメントルートを作成したディレクトリの「public」配下に設定しましょう。
※デフォルトは「htdocs」配下のため、今までの作成物を動作確認したい場合は設定を戻す必要があります。
• MAMP の場合：https://qiita.com/ndj/items/d837a9f20709bb5ec3d8
• XAMPP の場合：https://takro-blog.com/about-the-way-to-change-xampp-documentroot/

MVC 作成

処理の流れ

事前に今回プロジェクトの大まかな処理の流れを頭に入れて作り始めましょう。



MVC 作成

アクセス制御

public > .htaccess > IfModule

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php [QSA,L]
</IfModule>
```

まず public 配下の「.htaccess」というファイルを編集します。(頭のドットを忘れず)
左記のように、public 配下のファイルが見つからない場合に index.php を読むように設定 します。
「.htaccess」は Apache の設定をディレクトリ単位に行うことができます。
ここでは細か い記述の仕方は省略しますが、記述ミスが少しでもあると動作しなくなるため注意が必要です。

public > index.php

```
<?php
echo 'test';
```

記述できたら動作確認を行いましょう。
public 配下の「index.php」に適当な文字列を出力する構文を記述します。
ブラウザで、「localhost/index.php」と「localhost/xxx.php」(存在しないファイル) のそれぞれにアクセスした時、いずれも index.php に記載した文字列が表示されれば動作確認は完了です。(localhost 部分は環境によっては localhost:8888/xxx.php 等、ポート番号を記載)
※表示されない場合は「MAMP(or XAMPP) htaccess 有効 」などで調べてください。

MVC 作成

アクセス制御 2

public > index.php

```php
<?php
define('ROOT_PATH', str_replace('public', '', $_SERVER["DOCUMENT_ROOT"]));
$parse = parse_url($_SERVER["REQUEST_URI"]);
//ファイル名が省略されていた場合、index.phpを補填する
if(mb_substr($parse['path'], -1) === '/') {
    $parse['path'] .= $_SERVER["SCRIPT_NAME"];
}

require_once(ROOT_PATH.'Views'.$parse['path']);
```

続いて「index.php」ファイルの中身を編集します。
左記のように、入力 URL から Views 配下の読み込むファイルを判定します。
例えば「localhost/Players/xxx.php」とすれば
Views/Players 配下の「xxx.php」が読み込 ます。
Views 配下にディレクトリやファイルが存在しない場合、
require によるエラーが発生しま す。

MVC 作成

コントローラー

Controllers > PlayerController.php

```php
<?php
class PlayerController{
    public function __construct(){
    }

    public function index(){
        echo 'index';
    }
}
```

次にコントローラーを作成します。
Cotrollers 配下の「PlayerController.php」を編集します。
左記のようにクラスを作成し、コンストラクタと index メソッドを記述してください。
この段階ではデバッグ用の echo 以外は処理は空になってます。
※ここでは命名規則として Controller のファイル名は後続で作成する View と Model のものと揃えるようにしましょう。

MVC 作成

ビュー

Views > Players > index.php

```php
<?php
require_once(ROOT_PATH .'Controllers/PlayerController.php');
$player = new PlayerController();
$player->index();
?>
```

次にビューを作成します。
今回は Views 配下に Players というディレクトリを作り、配下にファイルを作成します。
左記のように require で任意のコントローラーを読み込み、インスタンスを生成後に先ほど作成した index メソッドを呼び出します。
「localhost/Players/index.php」にアクセスした場合、先ほどメソッド内で echo した文字列が表示されていれば OK です。

MVC 作成

コントローラーとビュー

Controllers > PlayerController.php

```php
<?php
class PlayerController{
    public function __construct(){
    }

    public function index(){
        $params = [
            'test' => 'index'
        ];
        return $params;
    }

}
```

Views > Players > index.php > ...

```php
<?php
require_once(ROOT_PATH . "Controllers/PlayerController.php");

$playerController = new PlayerController();
$params = $playerController->index();
?>

<p><?= $params['test'] ?></p>

```

続いて Controller と View 間での値受け渡しをしてみます。
Cotroller 側では index メソッド内で配列を作成し、返り値にセットしてみます。
View 側では上記返り値を受け取り画面へ表示しましょう。
「localhost/Players/index.php」にアクセスした場合、先ほどメソッド内でセットした文字列が表示されていれば OK です。

MVC 作成

モデル（接続情報）

database.php

```php
<?php
define('DB_HOST', 'localhost');//データベースのホスト名またはIPアドレス
define('DB_USER', 'root');//MySQLのユーザ名
define('DB_PASSWD', 'root'); //MySQLのパスワード
define('DB_NAME', 'worldcup');//データベース名
```

次にモデルを作成します。
先ずプロジェクト直下の「database.php」に データベース接続用の定数を左記のように記述しましょう。
※ユーザ名、パスワード、DB 名は自分の環境に合わせてください。

MVC 作成

モデル (接続用クラス)

Models > Db.php

```php
<?php
require_once(ROOT_PATH.'/database.php');

class Db {
    protected $dbh;

    public function __construct($dbh = null) {
        if(!$dbh) { //接続情報が存在しない場合
            try {
                $this->dbh = new PDO(
                    'mysql:dbname='.DB_NAME.
                    ';host='.DB_HOST, DB_USER, DB_PASSWD
                );
                //接続成功
            } catch (PDOException $e) {
                echo "接続失敗:".$e->getMessage()."\n";
                exit();
            }
        } else { //接続情報が存在する場合
            $this->dbh = $dbh;
        }
    }
}
```

続いて Models 配下のファイルを編集します。
先ずは DB 接続用の親クラスを作成します。
左記のように「Db.php」のコンストラクタに PDO 接続を記述します。
先ほど作成した「database.php」に定義した接続情報を require で読み込んで使用します。

MVC 作成

コントローラーとモデル

Controllers > PlayerController.php

```php
<?php
require_once(ROOT_PATH.'/Models/Db.php');

class PlayerController{
    protected $Db;

    public function __construct(){
        //モデルオブジェクトの生成
        $this->Db = new Db();
    }
```

次にコントローラーからモデルを呼び出します。
「PlayerController.php」に先ほど作成した「Db.php」を読み込みます。
コンストラクタにてインスタンスを生成し、「Db.php」のコンストラクタを呼び出します。
「localhost/Players/index.php」にアクセスし、接続に失敗していないかを確認しましょ う。

MVC 作成

コントローラーとモデル 2

Models > Player.php

```php
<?php
require_once(ROOT_PATH.'/Models/Db.php');

class Player extends Db {
    public function __construct($dbh = null) {
        parent::__construct($dbh);
    }
```

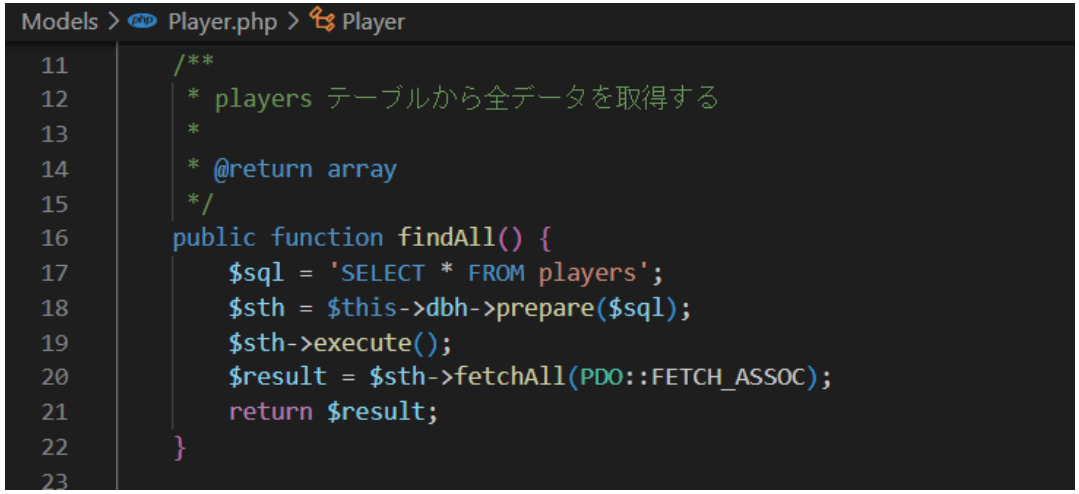
Controllers > PlayerController.php

```php
<?php
require_once(ROOT_PATH.'/Models/Player.php');

class PlayerController{
    protected $Player; //Playerモデル

    public function __construct(){
        //モデルオブジェクトの生成
        $this->Player = new Player();
    }
```

続いて接続用クラスを汎用的に使用するため、Db クラスを親とした子クラス「Player」を作成します。
左記のようにモデルではコンストラクタにて親クラスのコンストラクタを呼び出します。
コントローラーは先ほど Db クラスを呼び出していた箇所を Player クラスに変更します。
「localhost/Players/index.php」にアクセス し、接続に失敗していないかを確認しましょ う。

MVC 作成

MVC の連携（モデル）



それでは MVC を連携していきましょう。
ここでは players テーブルから選手データを取得するプログラムを作成してみます。
モデル「Player.php」に左記のように findAll というメソッドを記述します。
条件なしで SELECT を行い、結果を返す簡易な処理になります。
※メソッド横の Array は返り値の型を表します。

MVC 作成

MVC の連携（コントローラー）

Controllers > PlayerController.php > ...

```php
<?php
require_once(ROOT_PATH . "/Models/Player.php");

class PlayerController
{
    private $Player; // Playerモデル

    public function __construct() {
        // モデルオブジェクトの生成
        $this->Player = new Player();
    }

    /**
     * players テーブルから全データを取得する
     *
     * @return array
     */
    public function index() {
        $players = $this->Player->findAll();
        $params = [
            "players" => $players
        ];

        return $params;
    }

```

続いてコントローラーを編集します。
index メソッドにて Player クラスの findAll を呼び出し、
返り値を配列にセットして View に返します。

MVC 作成

MVC の連携（ビュー）

Views > Players > index.php

```
18      <h2>選手一覧</h2>
19      <table>
20          <tr>
21              <th>No</th>
22              <th>背番号</th>
23              <th>ポジション</th>
24              <th>名前</th>
25              <th>所属</th>
26              <th>誕生日</th>
27              <th>身長</th>
28              <th>体重</th>
29              <th></th>
30          </tr>
31          <?php foreach($params['players'] as $player): ?>
32              <tr>
33                  <td><?=$player['id'] ?></td>
34                  <td><?=$player['uniform_num'] ?></td>
35                  <td><?=$player['position'] ?></td>
36                  <td><?=$player['name'] ?></td>
37                  <td><?=$player['club'] ?></td>
38                  <td><?=$player['birth'] ?></td>
39                  <td><?=$player['height'] ?>cm</td>
40                  <td><?=$player['weight'] ?>kg</td>
41              </tr>
42              <?php endforeach; ?>
43      </table>
```

続いてビューを編集します。
コントローラーから返ってきたデータを左記のようにテーブル表示してみましょう。
「localhost/Players/index.php」にアクセスし、選手データが表示されていれば OK です。

MVC 作成

CSS、JS、画像など

Views > Players > index.php

```php
<?php
require_once(ROOT_PATH .'Controllers/PlayerController.php');
$player = new PlayerController();
$params = $player->index();
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>オブジェクト指向 - 選手一覧</title>
    <link rel="stylesheet" href="https://unpkg.com/ress/dist/ress.min.css">
    <link rel="stylesheet" type="text/css" href="/css/base.css">
    <link rel="stylesheet" type="text/css" href="/css/style.css">
</head>
<body>
```

※ここは飛ばしても良い

左記のように View 内のファイルに記述します。

頭の「/」はルートを表し、今回は冒頭で設定したドキュメントルート public になります。

ただし、画像赤枠内 1 行目はリセット CSS のため、URL での記述になっています。リセット CSS が何かわからない場合はこの機会に調べておきましょう。

public 配下のファイルのパスを示すことで CSS や JS や画像の読み込みができます。

今回は特にデザイン指定はありませんが見やすくするために最低限整えておきましょう。

MVC 作成

モデルのプロパティ

Models > Player.php

```php
<?php
require_once(ROOT_PATH.'/Models/Db.php');

class Player extends Db {
    private $table = 'players';
```

```php
    public function findAll():Array {
        $sql = 'SELECT * FROM '.$this->table;
        $sth = $this->dbh->prepare($sql);
        $sth->execute();
        $result = $sth->fetchAll(PDO::FETCH_ASSOC);
        return $result;
    }
```

※ここは飛ばしても良い

モデルには今後様々なデータ操作を記述する可能性があります。

プロパティにテーブル名を記述することでテーブル名の変更があった場合も共通して反映できます。

MVC 作成

リクエストパラメータ

選手一覧

No	背番号	ポジション	名前	所属	誕生日	身長	体重	
1	12	MF	ジュリオセザール	トロント（カナダ）	1979-09-03	186cm	79kg	詳細
2	3	MF	ジェフェルソン	ボタフォゴ	1983-01-23	188cm	80kg	詳細
3	22	MF	ビクトル	アトレチコ・ミネイロ（ブラジル）	1983-01-21	193cm	84kg	詳細
4	23	DF	マイコン	ローマ（イタリア）	1981-07-26	184cm	77kg	詳細
5	14	DF	マックスウェル	パリサンジェルマン（フランス）	1981-08-27	176cm	73kg	詳細
6	2	DF	アウベス	バルセロナ（スペイン）	1983-05-06	173cm	64kg	詳細
7	13	DF	ダンチ	Bミュンヘン（ドイツ）	1983-10-18	188cm	87kg	詳細
8	3	DF	チアゴシウバ	パリサンジェルマン（フランス）	1984-09-22	183cm	79kg	詳細
9	15	DF	エンリケ	ナポリ（イタリア）	1986-10-14	187cm	73kg	詳細
10	4	DF	ダビドルイス	チェルシー（イングランド）	1987-04-22	189cm	84kg	詳細
11	6	DF	マルセロ	Rマドリード（スペイン）	1988-05-12	172cm	73kg	詳細
12	5	MF	フェルナンジーニョ	マンチェスターC（イングランド）	1985-05-04	175cm	67kg	詳細
13	18	MF	エルナネス	インテルミラノ（イタリア）	1985-05-29	181cm	76kg	詳細
14	16	MF	ラミレス	チェルシー（イングランド）	1987-03-24	180cm	73kg	詳細
15	17	MF	ルイスグスタボ	ウォルフスブルク（ドイツ）	1987-07-23	187cm	80kg	詳細
16	8	MF	パウリーニョ	トットナム（イングランド）	1988-07-25	182cm	71kg	詳細
17	19	MF	ビリアン	チェルシー（イングランド）	1988-08-09	174cm	72kg	詳細
18	11	MF	オスカル	チェルシー（イングランド）	1991-09-09	180cm	66kg	詳細
19	9	FW	フレジ	フルミネンセ（ブラジル）	1983-10-03	185cm	90kg	詳細
20	7	FW	フッキ	ゼニト（ロシア）	1986-07-25	180cm	85kg	詳細

左記のように詳細リンクを用意します。
詳細ページには 1 選手のデータを表示します。
そのために画面から詳細リンクをクリックした際にリクエストパラメータ (ここでは選手 ID) を POST/GET 送信する必要があります。

MVC 作成

リクエストパラメータ 2

Controllers > PlayerController.php

```php
<?php
require_once(ROOT_PATH.'/Models/Player.php');

class PlayerController {
    private $request;
    private $Player;

    public function __construct(){
        //リクエストパラメータの取得
        $this->request['get'] = $_GET;
        $this->request['post'] = $_POST;

        //モデルオブジェクトの生成
        $this->Player = new Player();
    }
```

では詳細リンククリック後の処理を作成します。
まずは送信されたリクエストパラメータはコンストラクタで予め取得しておきます。

MVC 作成

リクエストパラメータ 3

Player.php

Models > Player.php > Player > findById

```php
22    public function findById($id)
23    {
24        $sql = 'SELECT * FROM players WHERE id = ' . $id;
25        $sth = $this->dbh->prepare($sql);
26        $sth->execute();
27        $result = $sth->fetch(PDO::FETCH_ASSOC);
28        return $result;
29    }
```

続いて詳細ページにモデルを作成します。
モデル「Player.php」に findById というメソッドを記述します。
ここでは引数として選手 ID を受け取り、それを条件として
選手データを取得します。

MVC 作成

リクエストパラメータ 4

```
41
42      /**
43       * プレイヤー詳細
44       *
45       * @return array
46       */
47      public function view() {
48          if (empty($this->request['get']['id'])) {
49              echo "指定のパラメーターが不正です。このページを表示できません。";
50              exit();
51          }
52
53          $player = $this->Player->findById($this->request['get']['id']);
54          $params = [
55              'player' => $player
56          ];
57
58          return $params;
59      }
60  }
61
```

続いてコントローラー「PlayerController.php」から先ほどモデル「Player.php」に記述した findById メソッドを呼び出します。
ちなみに id が送信されずに view が呼び出された場合、画面エラーを避けるためにもエラー処理をしておきましょう。

MVC 作成

リクエストパラメータ 5

■選手詳細

No	1
背番号	12
ポジション	MF
名前	ジュリオセザール
所属	トロント（カナダ）
誕生日	1979-09-03
身長	186cm
体重	79kg
編集 削除	

トップへ戻る

左記のようにデータが表示されれば OK です。
トップに戻れるようにリンクも用意しておきましょう。

MVC 作成

ページング

Models > Player.php > Player

```php
    /**
     * players テーブルから全データの件数を取得する
     *
     * @return int
     */
    public function countAll() {
        $sql = 'SELECT count(*) as count FROM players';
        $sth = $this->dbh->prepare($sql);
        $sth->execute();
        $count = $sth->fetchColumn();
        return $count;
    }
```

現状 1 ページに表示するデータ量が多いため、簡単なページング機能を追加していきます。
ここでは 1 ページあたり 20 件ごとに表示してみましょう。
まずはトータル件数を取得します。
モデル「Player.php」に左記のように countAll というメソッドを記述します。

MVC 作成

ページング 2

```
20      public function index() {
21          $players = $this->Player->findAll();
22          $players_count = $this->Player->countAll();
23          $params = [
24              'players' => $players,
25              'pages' => $players_count / 20
26          ];
27          return $params;
28      }
```

続いてコントローラー「PlayerController」の index メソッドにて Player クラスの countAll を呼び出します。
今回は 1 ページあたり 20 件ごとに表示するため、トータル件数を 20 で除算してページ数を算出します。
算出したページ数を配列にセットし、View へ 返します。

MVC 作成

ページング 3

```
43              <?php endforeach; ?>
44          </table>
45
46          <div class="paging">
47              <?php
48              for($i = 0; $i <= $params['pages']; $i++) {
49                  if(isset($_GET['page']) && $_GET['page'] == $i) {
50                      echo $i + 1;
51                  } else {
52                      echo "<a href='?page=".$i."'>".($i + 1)."</a>";
53                  }
54              }
55              ?>
56          </div>
57      </body>
58      </html>
```

続いてビュー「Players/index.php」のテーブル下部にページリンクを表示させます。
ここではリンクをクリックした際にページ番号をパラメータとして送るようにしています。

20	7	FW	フッキ	ゼニト（ロシア）	1986-07-25	180cm	85kg	詳細

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

MVC 作成

ページング 4

```
20      public function index() {
21          $page = 0;
22          if(isset($this->request['get']['page'])) {
23              $page = $this->request['get']['page'];
24          }
25
26          $players = $this->Player->findAll($page);
27          $players_count = $this->Player->countAll();
28          $params = [
29              'players' => $players,
30              'pages' => $players_count / 20
31          ];
32          return $params;
33      }
```

リクエストパラメータからページ番号を取得し、findAll を呼び出す際に引数として渡しま す。
初期表示用にページ番号は 0 で宣言します。

MVC 作成

ページング 5

```php
<?php
require_once(ROOT_PATH . '/Models/Db.php');

class Player extends Db
{
    const PLAYER_LIMIT = 20; // 一覧で表示するプレイヤーの件数

    private $table = 'players';

    public function __construct($dbh = null)
    {
        parent::__construct($dbh);
    }


    /**
     * @param int $page ページ番号
     * @return array
     *
     *
     * players テーブルから20件ごとデータを取得する
     */

    public function findAll($page = 0) {
        $sql = 'SELECT * FROM '. $this->table;
        $sql .= ' LIMIT ' . self::PLAYER_LIMIT . ' OFFSET ' . (20 * $page);
        $sth = $this->dbh->prepare($sql);
        $sth->execute();
        $result = $sth->fetchAll(PDO::FETCH_ASSOC);
        return $result;
    }
```

先ほどコントローラーから渡されたページ番号を条件に SQL を発行します。
ここでは LIMIT～OFFSET を利用します。
例えばページ番号 2 なら 21～40 の 20 件を取得することになります。

MVC 作成

複数モデルの使用

```
24      public function get_db_handler() {
25          return $this->dbh;
26      }
27  }
```

ここまではコントローラーとモデルを 1 対 1 として作成してきました。
またモデルはテーブルとも 1 対 1 の関係となっており、モデル内に複数テーブルの操作を記述してしまうのはコードを煩雑にしてしまいます。
しかしながら、画面によって 1 コントローラーにて複数モデルを使用したい場合が出てくるで しょう。
ここからは 1 コントローラーにて複数モデルを利用する際の記述例を見ていきましょう。
先ずは「Db.php」に DB ハンドラを返すメソッドを記述します。

MVC 作成

複数モデルの使用 2

```php
<?php
require_once(ROOT_PATH.'/Models/Player.php');
require_once(ROOT_PATH.'/Models/Goal.php');

class PlayerController {
    private $request; //リクエストパラメータ(GET,POST)
    private $Player; ///Playerモデル
    private $Goal; //Goalモデル

    public function __construct(){
        //リクエストパラメータの取得
        $this->request['get'] = $_GET;
        $this->request['post'] = $_POST;

        //モデルオブジェクトの生成
        $this->Player = new Player();
        //別モデルと連携
        $dbh = $this->Player->get_db_handler();
        $this->Goal = new Goal($dbh);
    }
```

例えば goals テーブルに関するモデル「Goal.php」を「PlayerController」から使用する場合、左記のように DB ハンドラを取得し、それを別モデルに渡すという形を取りま す。これは DB 接続自体は new Player() の時点で行っており多重接続することを避けるためで す。