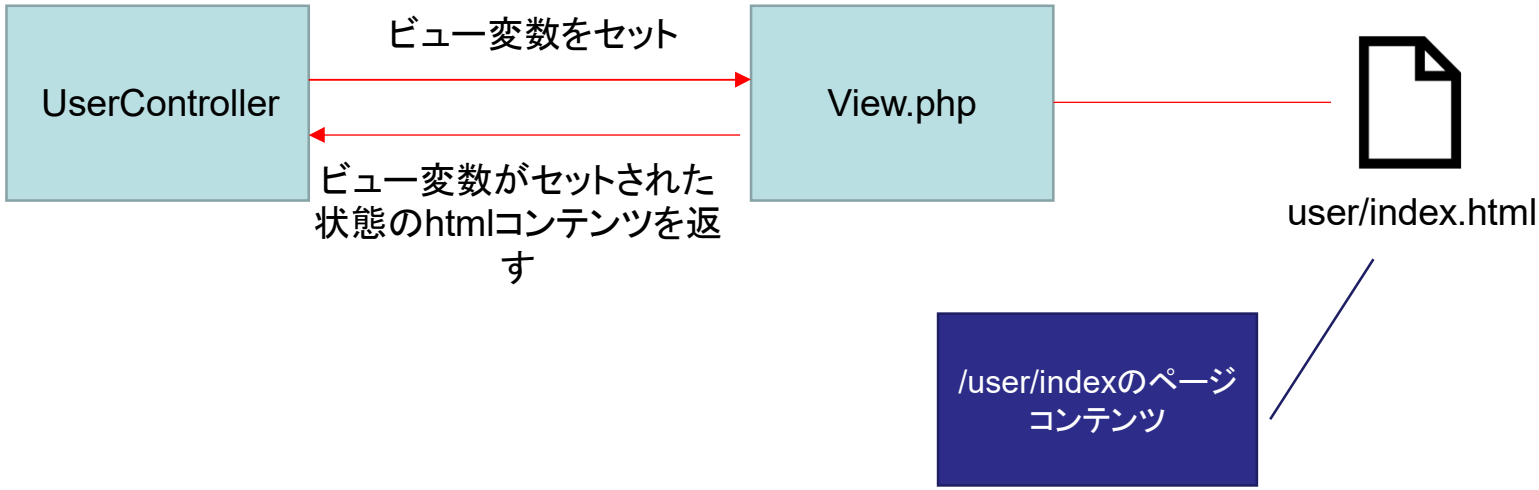
本ステップでおこなうこと

コントローラーとビュー(html)を分離します。



Viewクラスを作り、htmlファイルを分離する(1)

UserController.php

```
public function indexAction()
{
  $view = new View('user/index.html');
  $view->message='Hello';
  echo $view->render();
}
```

View.php

```
<h2>ユーザTOPページ</h2>
<p><?=$this->message?></p>
```

user/index.html

Viewクラスを作り、htmlファイルを分離する(2)

まずはrenderメソッドを実装してみます。

もっともシンプルな実装としては、Viewクラス内でincludeするだけです。

View.php

```
class View {
    public function render($file) {
        include($file);
    }
}
```

Viewクラスを作り、htmlファイルを分離する(3)

しかし、この実装では、View::render()をコールするのと同時に、user/index.htmlの内容が出力されてしまいます。
できれば、View::render()内で出力せずに、戻り値として受け取りたいところです。

UserController.php

```
public function sendMailAction()
{
  $view = new View('mail-body.txt');
  $view->message= 'お問い合わせありがとうございます。';
  $view->to = '山田太郎様';
  $mailBody = $view->render();
  $this->sendThankyouMail($mailBody);
}
```

Viewクラスを作り、htmlファイルを分離する(4)

そこで今回、View::render()内で使うのが、ob_*関数です。

★ob_start関数
→　出力バッファリングを開始する

★ob_get_contents関数
→　出力バッファの内容を得る

★ob_end()関数
→　出力バッファリングを終了する

Viewクラスを作り、htmlファイルを分離する(5)

以下の実験コードを実行すると…

```
<?php
ob_start();
echo 1, PHP_EOL;
echo 2, PHP_EOL;
$contents = ob_get_contents();
ob_end_clean();
echo 3, PHP_EOL;
echo $contents;
```

出力結果

```
3
1
2
```

Viewクラスを作り、htmlファイルを分離する(6)

結論、View::render()を以下のように実装すれば、
呼び出し元は戻り値として受け取ることができます。

View.php

```
class View {
    public function render($file) {
        ob_start();
        include($file);
        $html = ob_get_contents();
        ob_end_clean();
        return $html;
    }
}
```

ビュー変数をセットする

マジックメソッド__set、__getを使い、任意の変数をビューにセットできるようにします。

```
public function indexAction()
{
  $view = new View('user/index.html');
  $view->message='Hello';
  echo $view->render();
}
```

UserController.php

```
public function __set($name, $value)
{
    $this->datas[$name] = $value;
}

public function __get($name)
{
    return $this->datas[$name];
}
```

View.php

```
<h2>ユーザTOPページ</h2>
<p><?=$this->message?></p>
```

user/index.html

ビュー内で使えるメソッドを作る(1)

これから、HTMLファイルから呼び出せる、ビュー用のお助けクラス(＝ビューヘルパー)を作ります。
もっともシンプルな方法は、Viewクラス内にメソッドを定義することです。
これだけでも、HTMLファイルからメソッドを呼び出せます。

```
public function escape($value)
{
    return htmlspecialchars($value, ENT_QUOTES);
}
```

View.php


```
<h2>ユーザTOPページ</h2>
<p><?=$this->escape($this->message)?></p>
```

user/index.html

ビュー内で使えるメソッドを作る(2)

しかし、そのアプリケーションに固有の処理までViewクラスにもたせてしまうと、Viewクラスが汎用的ではなくなってしまいます。

```php
public function showErrors(?array $errors): string
{
    $list = '<ul class="error-area">';
    foreach ($errors as $error) {
        $list .= '<li>' . $error . '</li>';
    }
    $list .= '</ul>';
    return $list;
}
```

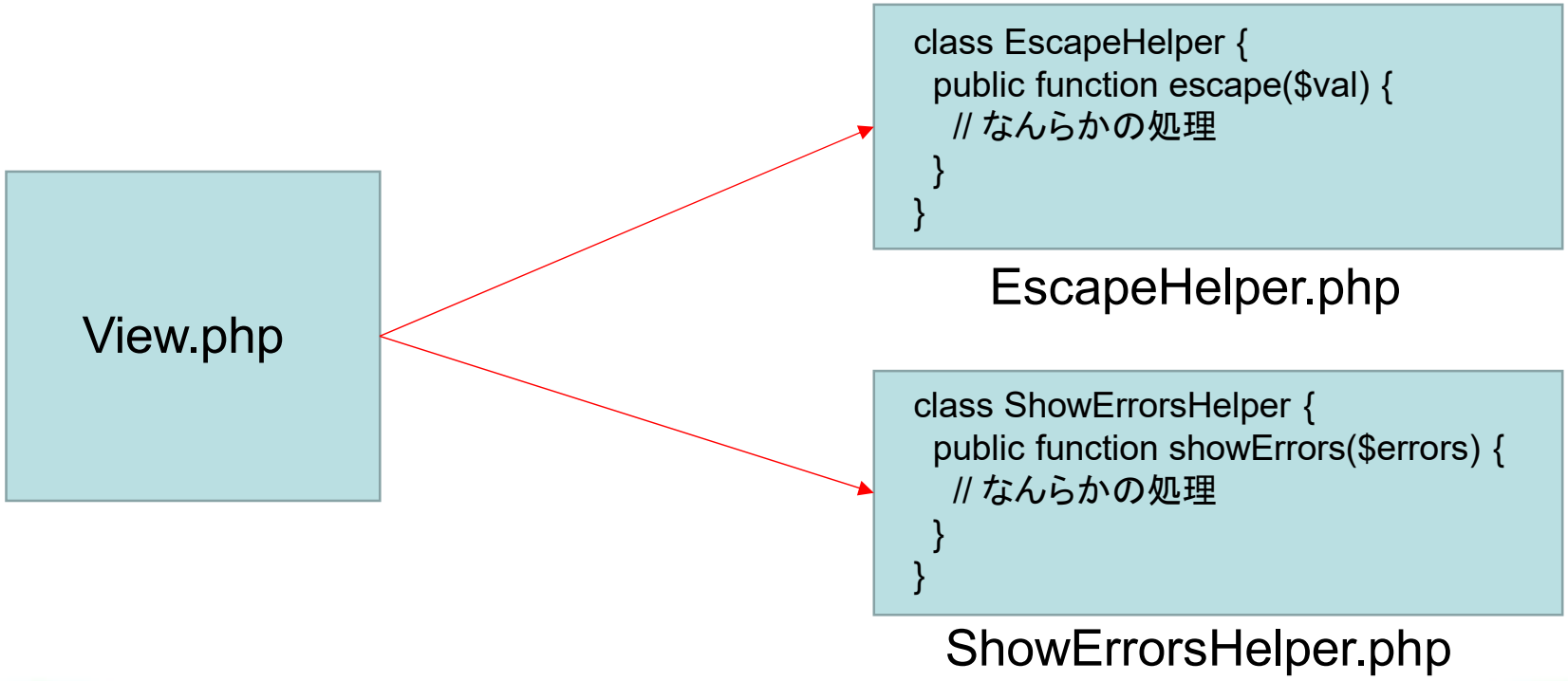
View.php

```php
<p><?=$this->showErrors($errors)?></p>
```

user/index.html

ビュー内で使えるメソッドを作る(3)

そこで、Viewクラスをより汎用的なものとするために、
View.phpとは別ファイルにメソッドを定義できるようにします。



別クラスにある任意のメソッドを呼び出す(1)

View.phpの外部にあるメソッドを呼び出すために、2つの命令を理解しましょう。
まずは、マジックメソッド__callです。

__call (string $name , array $arguments) : mixed

アクセス不能メソッド(存在しないメソッドなど)が呼び出されたときに呼び出されるマジックメソッド

別クラスにある任意のメソッドを呼び出す(2)

以下のプログラム例では、存在しないメソッドdoSomethingを呼び出しています。

```
class SomeClass
{
    public function __call(string $name, array $arguments)
    {
        echo 'メソッド', $name, 'がコールされました。', PHP_EOL;
        echo '引数は以下の通りです:', PHP_EOL;
        print_r($arguments);
    }
}

$someClass = new SomeClass();
$someClass->doSomething('Good', 'Morning');
```

★実行結果

```
メソッドdoSomethingがコールされました。
引数は以下の通りです:
Array
(
    [0] => Good
    [1] => Morning
)
```

別クラスにある任意のメソッドを呼び出す(3)

次に、call_user_func_array関数を理解しましょう。

call_user_func_array (callable $callback , array $param_arr) : mixed

引数を指定して、関数やメソッドをコールする。
別クラスのメソッドをコールしたいときは、引数$callbackを、

[$インスタンス名, 'メソッド名']

の形式で指定する。

別クラスにある任意のメソッドを呼び出す(4)

以下のプログラム例では、Viewクラスから、
EscapeHelper::escapeメソッドをコールしています。

```
class EscapeHelper
{
    public function escape(string $value): string
    {
        return htmlspecialchars($value, ENT_QUOTES);
    }
}

class View
{
    public function doEscape(string $value): string
    {
        $escapeHelper = new EscapeHelper();
        $methodDefinition = [$escapeHelper, 'escape'];
        $params = [$value];
        return call_user_func_array($methodDefinition, $params);
    }
}

$view = new View();
echo $view->doEscape('Tom & Jerry');
```

★実行結果

Tom & Jerry

別クラスにある任意のメソッドを呼び出す(5)

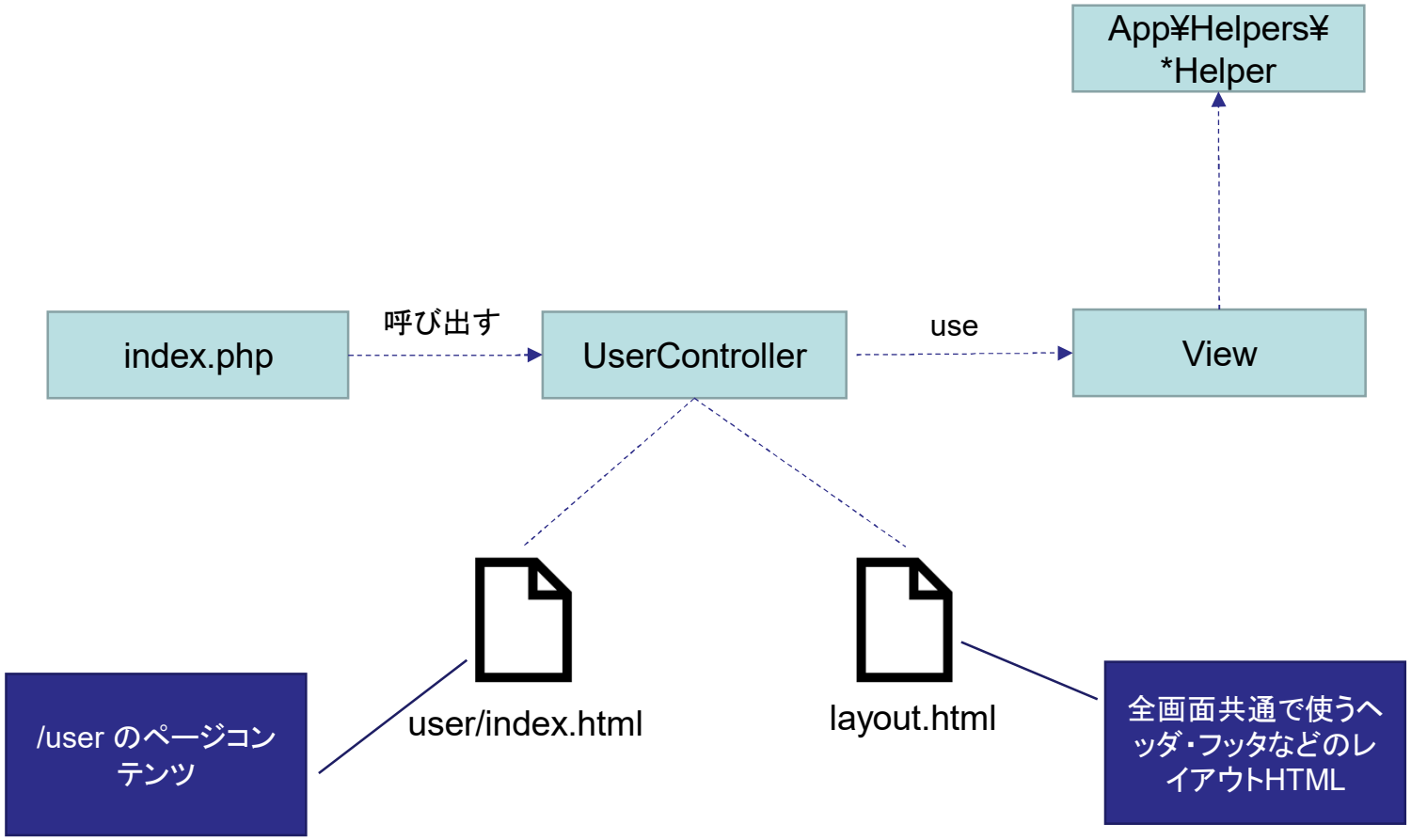
__callとcall_user_func_arrayを組み合わせることで、
外部クラスの任意のメソッドを呼び出すことができるようになります。

```
class EscapeHelper
{
    public function escape(string $value): string
    {
        return htmlspecialchars($value, ENT_QUOTES);
    }
}

class View
{
    public function __call(string $name, array $arguments)
    {
        $helperClass = ucfirst($name) . 'Helper';
        $helperInstance = new $helperClass();
        $callDefinition = [$helperInstance, $name];
        return call_user_func_array($callDefinition, $arguments);
    }
}

$view = new View();
echo $view->escape('Tom & Jerry');
```

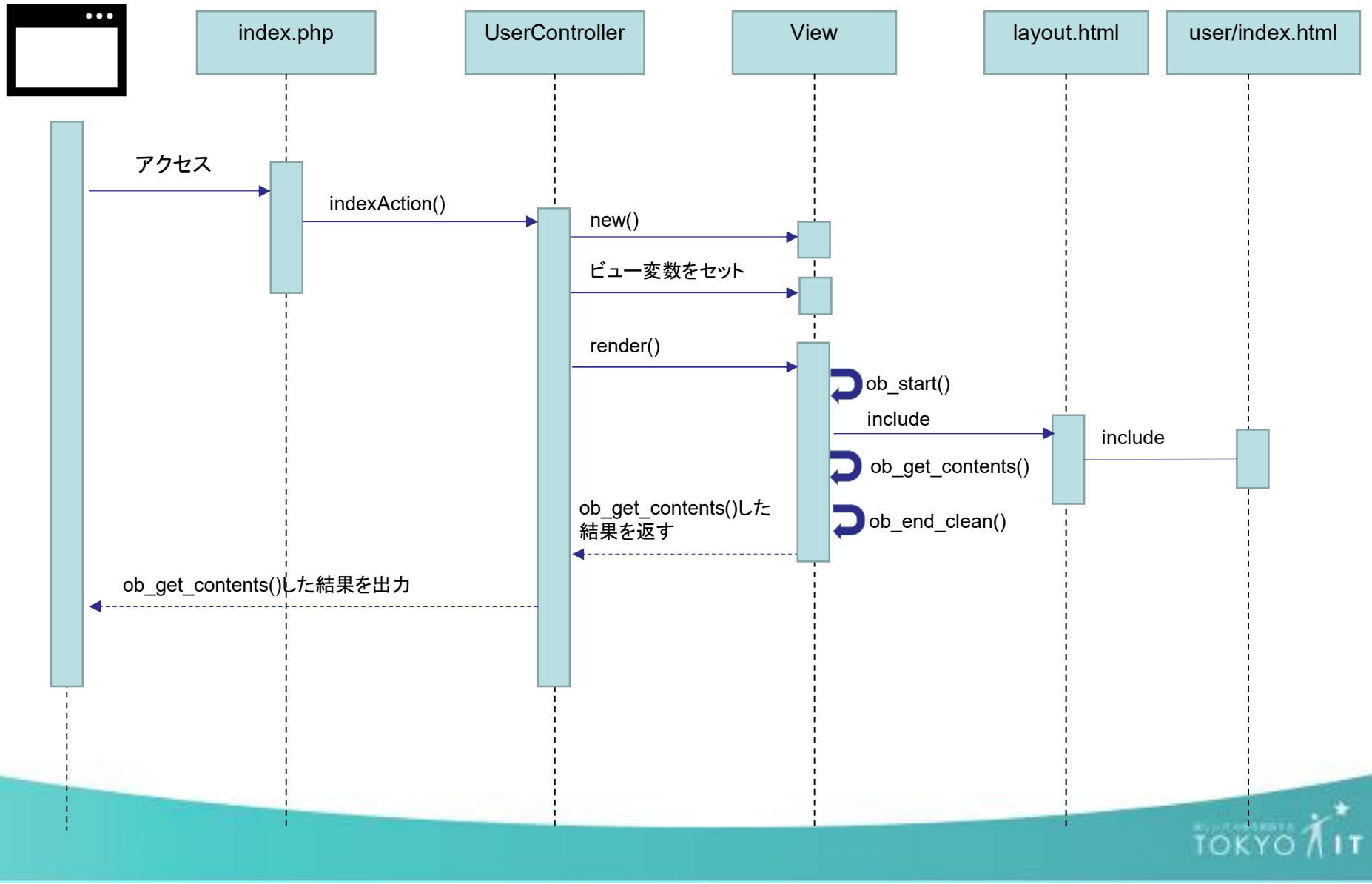
★実行結果

Tom & Jerry

本ステップのクラス構成



本ステップの処理の流れ



本ステップの変更ファイル一覧

●追加したファイル

- app/Libs/Core/View.php
  → ビューファイル(*.html)を読み込むクラス

- app/Helpers/*.php
  → ビューヘルパークラス群

- app/Modules/User/Layouts/layout*.html
  → ヘッダ・フッタなどのアプリ共通部分を記述したHTMLファイル

- app/Modules/User/Views/user/index.html
  → 画面別のHTMLファイル

- app/Modules/Common/Views/exception/index.html
  → step4で実装したエラーページ用のHTMLファイル

本ステップの変更ファイル一覧

●変更したファイル

- app/Modules/User/Controllers/UserController.php
  → indexActionから、user/index.htmlを読み込み、HTTPレスポンスとして出力する処理を追加した

参考情報

- PHP本格入門(上)
  「3-8 クラスの操作に自動で反応するメソッド - マジックメソッド」