Урок 3

Базовый шаблон

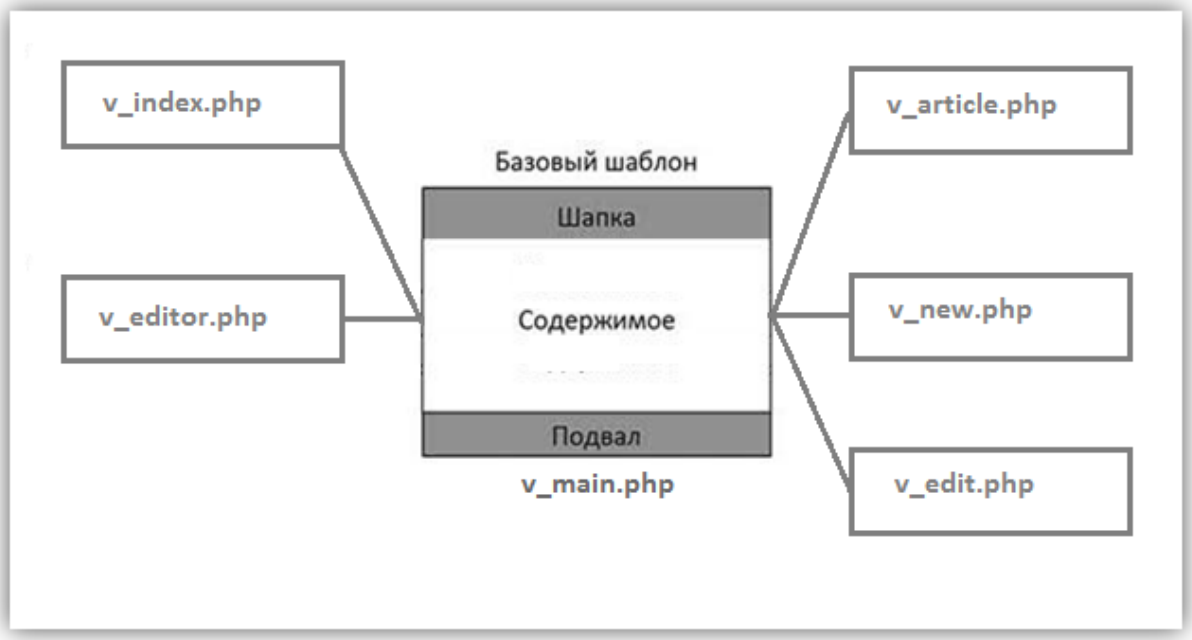
В этом уроке Вы узнаете:

- для чего нужен базовый шаблон;
- что такое вложенные шаблоны;
- что такое буферизация;
- универсальный способ работы с шаблонами.

1. Для чего нужен базовый шаблон

Наверняка, выполняя предыдущее домашнее задание, Вы заметили, что во всех шаблонах присутствовал одинаковый HTML-код. Скорее всего, в него входили только общие html-теги, вроде head, body и т.п., однако, на реальном сайте все страницы содержат одинаковые блоки, например, шапку и подвал. Получается, что все шаблоны будут содержать много повторяющегося кода и станут неудобными для редактирования.

Поэтому необходимо вынести все повторения в отдельный файл, который называется базовым шаблоном. Схематично это можно изобразить следующим образом:



В шаблона, отвечающих за конкретную страницу, мы оставляем только уникальный HTML-код, который с помощью PHP будем вставлять в середину базового шаблона.

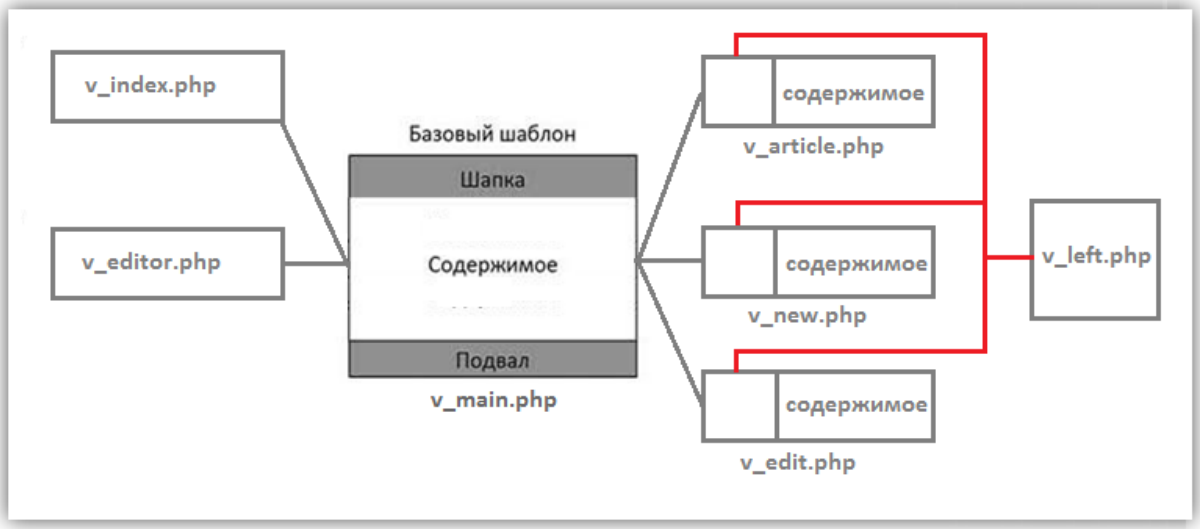
На более сложных сайтах могут возникать такие ситуации, при которых использование только одного базового шаблона не убирает весь повторяющийся код. Тогда появляются вложенные шаблоны.

2. Вложенные шаблоны

Представьте, что нам необходимо чуть-чуть усложнить блог – на страницах articles.php, add.php и edit.php сделать левый блок и выводить в него пять последних добавленных статей. Основная загадка заключается теперь в том, куда писать HTML-код этого блока. Если бы он выводился на всех страницах, то, однозначно, в базовый шаблон, если бы на одной – в шаблон конкретной страницы.

Но ведь в нашем случае список статей нужен на трёх страницах из пяти. Получается, что, с одной стороны, его в базовый шаблон положить нельзя, а с другой, если мы пропишем HTML-код левого блока для каждой страницы отдельно, - одинаковый код будет повторяться три раза.

Правильное решение – использование вложенного шаблона:



Т.е., мы выносим HTML-код левого блока в отдельный файл, который затем подключаем в шаблоны тех страниц, на которых его нужно отобразить.

В общем случае самая простая реализация подразумевает, что мы в основном шаблоне делаем include базового. Например, v_main может выглядеть следующим образом:

```
Код шапки сайта
include($path);      // $path - путь до вложенного шаблона
Код подвала сайта
```

А v_article так:

```
<div id="left">
    include('templates/v_left.php');
</div>
Код вывода одной статьи
```

И вот здесь кроется огромный минус такого прямого подхода. В контроллере мы, во-первых, не видим, какие шаблоны будут использоваться, а во вторых, не понимаем, какие переменные в какой шаблон пойдут. Поэтому придуман удобный и универсальный метод шаблонизации, который мы сейчас рассмотрим.

3. Буферизация в PHP

Данная тема является дополнительной, так как без неё невозможно реализовать самый удобный метод шаблонизации.

Буферизация – это отправка сообщений, которые мы выводит на экран, в буфер обмена.

Для того, чтобы начать вывод в буфер, существует специальная функция **ob_start()**, а для завершения буферизации и получения накопленного текста – функция **ob_get_clean()**.

Рассмотрим следующий фрагмент кода:

```
ob_start();
echo 'World';
$x = ob_get_clean();
echo "Hello, $x!";
```

Поскольку буферизация началась в первой строчке, echo 'World'; на экран ничего не выводит, а помещает данный текст в буфер обмена. В третьей строке буферизация завершается, а в переменной $x оказывается текст, накопленный в буфере. В итоге на экране мы увидим сообщение «Hello, World!», несмотря на то, что echo 'World'; было написано раньше.

4. Буферизированный способ подключения шаблонов

Для генерации строки с HTML-кодом шаблона создадим следующую функцию:

```
//
// Генерация шаблона.
//
function template($fileName, $vars = array())
{
    // Установка переменных для шаблона.
    foreach ($vars as $k => $v)
    {
        $$k = $v;
    }

    // Генерация HTML в строку.
    ob_start();
    include $fileName;
    return ob_get_clean();
}
```

где fileName – имя файла шаблона, vars – ассоциативный массив переменных для шаблона (имя – значение).

Общий смысл заключается в следующем: мы создаём функцию, которая принимает путь до шаблона и список переменных, которые нужно в него вставить, а возвращает сгенерированный HTML-код. Здесь и проявляется сила буферизации, так как, если бы её не было, команда **include** $fileName; сразу бы выводила шаблон на экран.

В итоге в контроллере у нас будет переменная с HTML-кодом определённого куска страницы.

Рассмотрим пример контроллера, использующего данную функцию:

```
<?php
include_once('model.php'
include_once('view.php'); // Считаем, что в этом файле лежит функция template

$title = 'Главная';
$articles = articles_all();

// Внутренний шаблон
$content = template ('v_index.php', array('articles' => $articles));

// Внешний шаблон
$page = template ('v_main.php',  array('title' => $title,
                                       'content' => $content));

// Вывод на экран
echo $page;
```

Теперь мы можем проследить полный путь генерации шаблона. В функцию template передаётся имя файла - 'v_index.php' и массив переменных - **array**('articles' => $articles)). В момент вызова оператора include в шаблоне будут видны только те переменные, которые объявлены в функции. Поэтому в самом её начале мы запускаем цикл, который разбирает массив и создаёт переменные по следующему правилу: ключ массива становится именем переменной, а значение по ключу – её значением.

Обратите внимание на конструкцию $$k. Она означает, что мы создаём переменную, имя которой совпадает со значением $k.

В итоге, будет сформирована переменная $articles, которую и увидит шаблон при вызове **include** $fileName. Сгенерированный код возвращается в контроллер в $template. Затем точно таким же способом происходит генерация внешнего шаблона.

На первый взгляд может показаться, что мы сильно усложнили систему. Однако, на самом деле, функцией template можно пользоваться даже не зная, как она устроена. Данная функция создаётся один раз, а затем мы просто пользуемся теми удобными возможностями, которые она даёт.

Самое главное, что данный способ подключения шаблонов имеет три важнейших плюса:

1) в контроллере мы сразу видим список всех шаблонов, используемых для построения данной страницы;
2) каждый шаблон видит только те переменные, которые мы ему передаём. Т.е., не может возникнуть ситуации конфликта имён переменных в разных шаблонах;
3) до самого последнего момента весь HTML-код не выводится на экран, а просто хранится в переменных. Благодаря этому, мы может не опасаться ошибок, вроде «header already sent», которые возникают в том случае если мы пытаемся отправлять заголовки, но уже вывели что-либо на экран.

Буферизированный способ является очень популярным решением и используется во многих современных фреймворках. Поэтому обязательно постарайтесь прочувствовать его плюсы.

Самоконтроль

- ✓ Зачем нужен базовый шаблон
- ✓ Что такое вложенные шаблоны
- ✓ Когда имеет смысл использовать вложенные шаблоны
- ✓ Что такое буферизация
- ✓ ob_start()
- ✓ ob_get_clean()
- ✓ Что означает конструкция $$
- ✓ Чем удобен буферизированный вывод шаблона

Домашнее задание

Выделить базовый шаблон блога. Все шаблоны должны подключаться буферизированным способом.