Глава 16. Нетривиальные приемы применения тем

Жасин Луиси

Слой тем в Drupal примечателен своей гибкостью. В предыдущей главе мы познакомились с основами создания тем: поработали с файлами .info, шаблонами и функциями тем. Однако в творческой работе этих инструментов порой не хватает, требуется нечто большее. Именно здесь проходит граница между разработчиками интерфейса и серверных приложений.

Закончив чтение данной главы, вы научитесь работать с переменными в функциях предварительной обработки, настраивать формы и применять новый прикладной программный интерфейс визуализации (render API). Также будут рассмотрены детали работы с CSS-файлами, основы создания подтем и базовые правила получения жизнеспособных тем в Drupal. Постепенно вы станете настоящим экспертом по этим вопросам.

Доступные переменные в слое тем

Познакомившись со слоем тем, вы быстро обнаружите, что в зависимости от того, с каким элементом вам приходится работать, появляются разные переменные. Кроме того, окажется, что различные шаблоны и функции тем используют далеко не все доступные переменные, причем исчерпывающая документация по данному вопросу может отсутствовать. Соответственно вам нужно будет самостоятельно выводить на экран содержимое массивов.

При помощи PHP это можно сделать разными способами. Чаще всего для этой цели применяется функция `print_r()`. Существуют также функции `var_dump()`, `get_defined_vars()` и собственная Drupal-функция `debug()`. Они прекрасно работают с небольшими массивами, но в Drupal иногда приходится иметь дело с настоящими монстрами. При написании интерфейса необходимость постоянно вызывать эти функции раздражает, если не сказать больше. К счастью, благодаря модулю Devel (http://drupal.org/project/devel) и библиотеке Krumo стал доступен простой и компактный механизм вывода содержимого массивов. Установив модуль Devel, вы в числе прочего получаете доступ к таким функциям, как `dpm()` и `kpr()`.

При работе с шаблонами и функциями предварительной обработки вывод переменных `$variables` обычно производится при помощи функции `dsm()` или `dsm()`. Для примера попробуйте добавить строку `<?php dpm($variables); ?>` в начало файла node.tpl.php, как показано на рис. 16.1.

Рис. 16.1. Результат добавления строки <?php dpm($variables); ?> в файл node.tpl.php

В функции `dpm()` хорошо то, что массив аккуратно выводится с применением переменной `$messages` в файле page.tpl.php, то есть там, где находятся сообщения о состоянии системы. Как показано на рис. 16.2, щелкнув на заголовке, можно по очереди раскрыть все содержимое массива.

Внутри шаблонов эти переменные становятся доступны как переменные верхнего уровня. Это сделано для удобства разработчиков тем. Скажем, вместо вывода `$variables['status']`

достаточно вывести переменную `$status` в шаблоне. Если же вы работаете с функциями изнутри, используйте переменную `$variables['status']`.

Рис. 16.2. Массив в развернутом виде, выведенный с помощью функции dpm()

Модуль Theme Developer

Начиная работать с Drupal, нужно выяснить, где находится код и что вам следует переопределить первым делом. В этом вам может помочь прекрасный модуль Theme Developer (http://drupal.org/project/devel_themer). После его подключения в нижнем левом углу страницы появляется флажок. Его установка приводит к появлению в верхнем правом углу экрана полупрозрачного окна. Вы можете менять его размер и положение в пространстве простым перетаскиванием мышью. Выделите щелчком любой элемент страницы, и в окне появится вся требуемая информация, как показано на рис. 16.3.

К примеру, после выделения узла в окне появляется следующая информация:

- родительские функции и шаблоны, влияющие на вид элемента;
- шаблон или вариант хука темы (вероятный);
- используемые функции предварительной и обычной обработки;
- список доступных переменных.

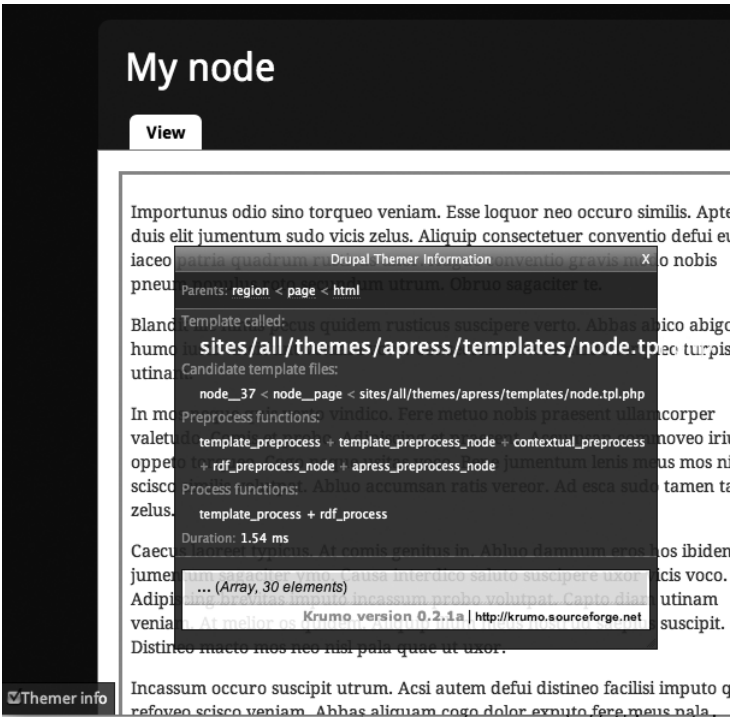

Рис. 16.3. В окне Theme Developer выводится информация о выбранном элементе (в данном случае — об узле)

Функции предварительной и обычной обработки

Лучшими друзьями разработчика тем являются функции предварительной обработки. Во многих случаях они способны сделать вашу жизнь проще, код — эффективнее, а файлы шаблонов — простыми и лаконичными. Если вы до сих пор ими не пользовались, думая, что они вам не нужны или боясь слишком глубоко погружаться в дебри PHP, вы много потеряли. Но я надеюсь вам помочь.

Для вас уже не является секретом предназначение шаблонов, которые в основном обеспечивают вас разметкой и показывают переменные. А теперь представьте, что вы хотите изменить эти переменные или добавить собственные. Скорее всего, вы решите, что нужно создать шаблон и отредактировать его нужным образом, но на самом деле это неверный путь.

Для указанной цели придуманы функции предварительной обработки. С их помощью вы, по сути, сообщаете Drupal: «Погоди! Я хочу внести изменения в эти данные до начала их визуализации». Это как редактор, читающий статью перед тем, как она будет опубликована. По определению, «предварительной обработкой» называется этап, предваряющий визуализацию шаблонов. Появившиеся в Drupal 7 функции обычной обработки служат той же цели, просто применяются позже.

Хорошим примером применения в Drupal функций обоих видов являются переменные `$classes_array` и `$classes variables`. В показанной в листинге 16.1 функции `template_preprocess()`, реализующей процесс предварительной обработки, предлагаемый по умолчанию, первой вызываемой функцией предварительной обработки инициализируется переменная `$classes_array` (см. http://api.drupal.org/api/function/template_preprocess/7).

Листинг 16.1. Фрагмент функции template_preprocess(), в котором определяется переменная $classes_array

```
<?php
function template_preprocess(&$variables, $hook) {
  // Инициализация атрибута class для текущего хука.
  $variables['classes_array'] = array(drupal_html_class($hook));
}
?>
```

Первый шаг обеспечивает вас классом, указывающим, какой именно хук будет использоваться. К примеру, если вызвать эту функцию предварительной обработки для узла, код добавит в массив класс node. После вызова функции смогут запускаться все модули и функции, добавляя или изменяя любые переменные.

Затем следует модуль Node, реализующий функцию `template_preprocess_node()` (см. http://api.drupal.org/api/function/template_preprocess_node/7). Как видно из листинга 16.2, в массив добавлено несколько классов.

Листинг 16.2. Фрагмент функции template_preprocess_node(), в котором в переменную $classes_array добавляются дополнительные классы

```
<?php
function template_preprocess_node(&$variables) {
  // Собираем классы узлов.
  $variables['classes_array'][] = drupal_html_class('node-' . $node->type);
  if ($variables['promote']) {
    $variables['classes_array'][] = 'node-promoted';
  }
  if ($variables['sticky']) {
    $variables['classes_array'][] = 'node-sticky';
  }
  if (!$variables['status']) {
    $variables['classes_array'][] = 'node-unpublished';
  }
  if ($variables['teaser']) {
    $variables['classes_array'][] = 'node-teaser';
  }
  if (isset($variables['preview'])) {
    $variables['classes_array'][] = 'node-preview';
  }
}
?>
```

И снова после вызова функции `template_preprocess_node()` все модули и темы получают возможность реализовать собственные версии, внести любые изменения и добавления. А после завершения функций предварительной обработки запускаются обычные функции. В ядре Drupal существует всего два процесса для узлов: реализуемая по умолчанию функция `template_process()` и функция `rdf_process()`, реализуемая модулем RDF.

В функции `template_process()` после того, как все модули и темы внесут свои изменения, появляется переменная `$classes`. Она содержит строковую версию всех классов, представленных в массиве `$classes_array`. Переменная `$classes` выводится в атрибуте `class` контейнера `<div>` в файле node.tpl.php, как показано в листинге 16.3.

Листинг 16.3. Фрагмент функции template_process(), в котором из переменной $classes_array создается переменная $classes

```
<?php
function template_process(&$variables, $hook) {
  // Выравнивание классов.
  $variables['classes'] = implode(' ', $variables['classes_array']);
}
?>
```

Листинги с 16.1 по 16.3 иллюстрируют не только гибкость и мощь функций предварительной и обычной обработки в Drupal, но и очередность их появления. Важно понять, что эти переменные не существуют вне слоя тем. А реализуя функции предварительной и обычной обработки, вы можете добавлять, модифицировать и удалять темы нужным вам образом. О том, как это делается, мы подробно поговорим далее.

Функции предварительной обработки позволяют вынести логические операции за рамки шаблонов. Это упрощает шаблоны, делая их более лаконичными, и заодно упрощает процедуру редактирования тем. Вы можете вносить множество изменений, например воздействовать на классы или модифицировать существующие переменные, и при этом вам не потребуется менять шаблоны — достаточно будет нескольких строк кода.

Хуки предварительной и обычной обработки

Функции предварительной обработки реализуются путем присвоения обычным функциям определенных имен. Пример принятых стандартов именования показан в листинге 16.4.

Листинг 16.4. Стандарты именования для хуков предварительной и обычной обработки

```
<?php
/**
* Реализует template_preprocess_THEMEHOOK().
*/
function HOOK_preprocess_THEMEHOOK(&$variables) {
  // Здесь следуют изменения.
}

/**
* Реализует template_process_THEMEHOOK().
*/
function HOOK_process_THEMEHOOK(&$variables) {
  // Здесь следуют изменения.
}
```

При именовании таких функций следует учитывать четыре момента:

1. Предлагаемая по умолчанию реализация хука, обычно созданная модулем, имеет в названии слово «template». В остальных случаях хук заменяется системным именем реализующего его модуля или темы.
2. На какую стадию процесса вы хотите воздействовать? Существуют два варианта: предварительная стадия, которая запускается в первую очередь, и обычная, запускаемая после выполнения всех функций предварительной обработки.
3. Хук темы совпадает со своим определением в функции `hook_theme()` и в конце концов выводится с именем либо функции темы, либо файла шаблона.
4. Параметр `&$variables` содержит данные, позволяющие функции темы или шаблону осуществить визуализацию. Так как функции предварительной обработки запускаются до визуализации шаблонов, вы можете вносить в них любые изменения и дополнения.

ВНИМАНИЕ

По умолчанию хуками предварительной обработки могут пользоваться только хуки темы, в явном виде определенные в функции hook_theme(). К примеру, это может быть hook_preprocess_node(), в то время как хук hook_preprocess_node__article() уже работать не будет. Ведь node__article — это всего лишь вариант хука темы.

Реализации по умолчанию

В листинге 16.5 показано, как выглядит реализация процесса предварительной обработки для хука темы, предлагаемого по умолчанию, на примере функции `template_preprocess_node()`,

создающей переменную для файла шаблона node.tpl.php. Эта функция помещена в node.module вместе с реализацией `hook_theme()` — `node_theme()`, где в качестве хука темы определяется «node».

Листинг 16.5. Стандарты именования для предлагаемых по умолчанию реализаций хуков предварительной и обычной обработки

```
<?php
function template_preprocess_node(&$variables) {
  // Здесь следуют изменения.
  // См. http://api.drupal.org/api/function/template_preprocess_node/7.
}

function template_process_node(&$variables) {
  // Здесь следуют изменения.
  // См. http://api.drupal.org/api/function/template_process_node/7.
}
```

СОВЕТ

Чтобы понять, как создаются переменные, исследуйте предлагаемые по умолчанию реализации на сайте http://api.drupal.org.

Реализации через темы и модули

Как модули, так и темы используют функции предварительной обработки одним и тем же способом. При этом каждый хук темы может иметь на этой стадии множество реализаций, полученных как при помощи модулей, так и при помощи тем. Это может стать причиной конфликта версий, поэтому следует четко придерживаться порядка запуска данных функций. Сначала запускаются функции предварительной обработки, сгенерированные модулями, и только потом — полученные при помощи тем. У последних также существует своя иерархия, так как в первую очередь запускаются версии базовых тем, и только потом — версии дочерних тем.

Функции предварительной обработки, реализованные ядром Drupal и модулями, находятся в разных файлах, например в modulename.module или theme.inc и во многих других. А вот функции предварительной обработки, реализованные темами, всегда оказываются в файле template.php.

В качестве примера реализуем функцию предварительной обработки для хука темы узла, взяв тему «dgd7». Как показано в листинге 16.6, вы просто помещаете функцию в файл template.php, начав с имени темы (реализующей хук), за ним следует суффикс `_preprocess_` и хук темы, который в нашем случае называется «node». Напоследок в переменную `&$variables` передается параметр по ссылке (именно передачу параметра по ссылке обозначает знак `&` перед знаком `$`).

Листинг 16.6. Реализация функции template_preprocess_node() через тему

```
<?php
/**
* Реализует template_process_node().
*/
function dgd7_preprocess_node(&$variables) {
  // Здесь идут изменения.
}
```

Представленного в листинге 16.6 кода с быстрой очисткой кэша достаточно для того, чтобы система Drupal перед визуализацией узла запустила вашу функцию предварительной обработки. Итак, все готово для работы!

Содержимое массива $variables

Содержимое массива $variables свое для каждого хука темы; более того, даже для одного хука оно будет зависеть от ряда других факторов, например от режима представления или роли пользователя.

Первое, что следует сделать после создания функции, это вывести содержимое данного массива и узнать, с чем вам предстоит работать. Как объяснялось в предыдущей главе, это можно сделать при помощи функции `dpm()`. Именно ее применение демонстрирует листинг 16.7.

Листинг 16.7. Вывод переменных с целью отладки

```
<?php
/**
* Реализуется template_preprocess_node().
*/
function dgd7_preprocess_node(&$variables) {
  dpm($variables);
}
```

ВНИМАНИЕ

Функции отладки используются только на стадии разработки.

Функции предварительной обработки в действии

Функции предварительной обработки позволяют модифицировать столько всего, что на рассмотрение всех возможных вариантов не хватит целой книги. Поэтому теперь, когда вы знаете, как настроить такую функцию и как увидеть значения доступных переменных, мы на практических примерах рассмотрим процедуру модификации.

Добавление классов к контейнерам шаблона

В теме DGD7 на сайте http://definitivedrupal.org области заголовка, врезки и нижнего колонтитула черные, а область контента — белая. Чтобы упростить добавление стиля к содержимому областей, добавим пару вспомогательных классов к контейнеру региона. Для этого следует реализовать функцию предварительной обработки для хука темы региона в файле template.php, как показано в листинге 16.8.

Листинг 16.8. Добавление классов к контейнеру . Переменная $classes_array используется в функции template_preprocess_region()

```
<?php
/**
* Реализуется template_preprocess_region().
*/
function dgd7_preprocess_region(&$variables) {
  $region = $variables['region'];
  // Областям врезки и контента нужен хороший класс. Не следует применять
  // такие идентификаторы, как #main или #main-контейнер, для стиля контента.
  if (in_array($region, array('sidebar_first',
    'sidebar_second', 'content'))) {
      $variables['classes_array'][] = 'main';
  }
  // Добавляем класс "clearfix" к регионам для очистки плавающих регионов.
  if (in_array($region, array('footer', 'help', 'highlight'))) {
    $variables['classes_array'][] = 'clearfix';
  }
```

```
  // Добавляем класс "outer" к более темным регионам.
  if (in_array($region, array('header', 'footer', 'sidebar_first',
    'sidebar_second'))) {
      $variables['classes_array'][] = 'outer';
  }
}
```

При обработке массив `$variables['classes_array']` превращается в переменную `$class`. При этом автоматически преобразуются класс или классы, добавленные на этапе предварительной обработки. То есть таким образом вы добавили класс к контейнеру `<div>` региона.

Аналогичного результата можно добиться через файлы шаблонов, но это более длинный путь. Вам потребуется добавить алгоритм в каждый из используемых шаблонов. При этом переопределение файлов осуществляется даже в случаях, когда менять разметку не требуется. Плюс изменения вручную вносятся в каждый шаблон для региона, что намного снижает эффективность работы. Поэтому лучше действовать по схеме, показанной в листинге 16.9.

Листинг 16.9. Добавление классов в функции предварительной обработки значительно повышает эффективность CSS-кода

```
/* Используются классы и ID, представленные по умолчанию. */
#header fieldset,
#footer fieldset,
.sidebar fieldset {
  border-color: #333
}

/* Используется класс, добавленный в листинге 16.8, что более эффективно. */
.outer fieldset {
  border-color: #333;
}
```

СОВЕТ

В данном примере меняются классы в шаблоне региона, но данную технику можно применять и к другим основным шаблонам, в том числе html.tpl.php, block.tpl.php, node.tpl.php и comment.tpl.php, достаточно воспользоваться соответствующими функциями предварительной обработки.

Внесение изменений в узлы

Листинг 16.10 демонстрирует следующие изменения:

1. Заголовок страницы выводится в файле page.tpl.php. Заголовок узла обычно выводится в файле node.tpl.php, когда он просматривается в режиме анонсов, и поэтому по умолчанию для него используется тег `<h2>`. Внутрь таких тегов помещается, как правило, и остальной контент из тела узла. Добавлением класса для выделения заголовка узла можно упростить дизайн. В листинге 16.10 этой цели служит массив `$title_attributes_array`.
2. Если форма добавления комментариев находится непосредственно под ссылкой на узел, дополнительная ссылка Add new comment теряет смысл. В листинге 16.10 она скрыта при помощи функции `hide()`, которая подробно рассматривается чуть позже.
3. Часто требуется, чтобы дизайн анонса узла отличался от дизайна всей страницы. В листинге 16.10 переменная `$variables['teaser']` уменьшает содержимое переменной `$submitted` и обрезает заголовок до 70 символов.

Листинг 16.10. Редактирование вида контента узла на стадии предварительной обработки

```
<?php
/**
* Реализуем template_preprocess_node().
```

продолжение ⇩

Листинг 16.10 (*продолжение*)

```
*/
  function dgd7_preprocess_node(&$variables) {
  // Присваиваем лучший класс тегу <h2>, содержащему заголовок анонса.
  $variables['title_attributes_array']['class'][] = 'node-title';

  // При наличии снизу формы скрываем ссылку "Add new comment".
  if (!empty($variables['content']['comments']['comment_form'])) {
    hide($variables['content']['links']['comment']);
  }

  // Вносим изменения, которые будут отличать режим анонса.
  if ($variables['teaser']) {
    // Не отображаем сведения об авторе и дате добавления.
    $variables['display_submitted'] = FALSE;
    // Обрезаем заголовок узла и добавляем эллипс.
    $variables['title'] = truncate_utf8($variables['title'],
      70, TRUE, TRUE);
  }
}
```

Ссылка на редактирование фото пользователя

Как вы, вероятно, уже заметили, массив `$variables` содержит множество переменных. И на их основе легко создавать новые переменные. Как вы знаете, адрес страницы редактирования пользовательского профиля — user/UID/edit. Сведения из переменной `$variables` позволяют определить, является ли пользователь, просматривающий профиль, его владельцем. После этого можно легко создать переменную, содержащую ссылку, которая дает пользователю возможность перейти к редактированию своего фото с любой страницы сайта. Для этого мы реализуем функцию `template_preprocess_user_picture()`, как показано в листинге 16.11. Затем ссылку можно будет вывести в соответствующем шаблоне user-picture.tpl.php, как показано в листинге 16.12.

Листинг 16.11. Создание новой переменной в файле user-picture.tpl.php при помощи функции template_preprocess_user_picture()

```
<?php
/**
* Реализуем template_preprocess_user_picture().
* - Добавляем ссылку "change picture" под фото пользователя.
*/
function dgd7_preprocess_user_picture(&$vars) {
  // Создаем переменную с пустой строкой, чтобы избежать извещений PHP
  // при выводе переменной
  $vars['edit_picture'] = '';
  // Объект account содержит сведения о пользователе, фото которого
  // обрабатывается. Сравним их с id объекта user, который представляет
  // авторизованного в данный момент пользователя.
  if ($vars['account']->uid == $vars['user']->uid) {
    // Создаем переменную со ссылкой на профиль пользователя с классом
    // "change-user-picture" для определения CSS-стиля.
    $vars['edit_picture'] = l('Change picture',
      'user/' . $vars['account']->uid . '/edit',
      array(
        'fragment' => 'edit-picture',
        'attributes' => array('class' => array('change-user-picture')),
      )
    );
  }
}
```

Листинг 16.12. Вставка новой переменной в файл user-picture.tpl.php, на основе которого переопределяется тема

```
<?php if ($user_picture): ?>
  <div class="user-picture">
    <?php print $user_picture; ?>

    <?php print $edit_picture; ?>

  </div>
<?php endif; ?>
```

Render API

Понятие массива визуализации

Многие переменные в файлах шаблонов выводятся непосредственно, в то время как для вывода других требуется функция `render()`. Массивами визуализации (render arrays) называют структурированные массивы, содержащие вложенные данные и другую информацию, которую Drupal использует для превращения шаблонов в HTML-код при помощи прикладного программного интерфейса визуализации (Render API). Именно массивы визуализации выводятся с использованием функции `render()`.

Если рассмотреть внимательно файл page.tpl.php, легко заметить, что таким способом выводятся все регионы. Каждый регион представляет собой элемент (еще один массив), вложенный в массив `$page`. Показанный в листинге 16.13 код — это все, что нужно для визуализации любого региона. Каждая функция `render()` возвращает полностью отформатированный HTML-код для всего содержимого массива визуализации.

Листинг 16.13. Вывод регионов в файле page.tpl.php с помощью функции render()

```
<?php print render($page['sidebar_first']); ?>
```

В предыдущих версиях Drupal нужно было вставлять директиву `<?php print $sidebar_first; ?>`, содержавшую готовую для отображения полностью отформатированную HTML-строку. Но такой подход отличался намного меньшей гибкостью.

В Drupal 7 эти переменные отправляются в шаблоны в виде аккуратно структурированных массивов. Вместо большого объема HTML-кода вы получили массив сведений о контенте, вплоть до атрибутов конкретных ссылок. Это серьезно упрощает жизнь, так как теперь вы можете вносить любые изменения буквально в последнюю секунду перед визуализацией.

Чтобы узнать содержимое массива, используйте функцию `dpm()` модуля Devel. Она выводит данные в файл page.tpl.php: `<?php dpm($page['sidebar_first']); ?>`. Как видно из рис. 16.4, внутри массива существуют два верхних уровня визуализации элементов — блок Search form и блок Navigation, которые выводятся на первой врезке.

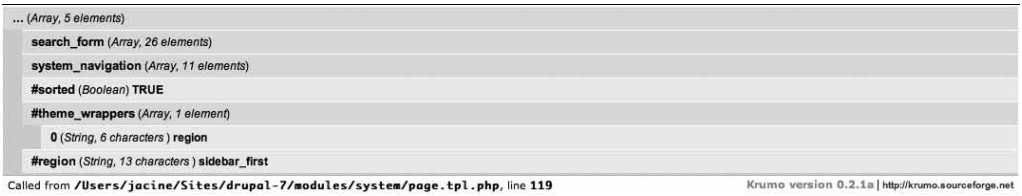

Рис. 16.4. Содержимое массива визуализации $page['sidebar_first'], выведенное из файла page.tpl.php с помощью функции dpm()