Данил Костров

Применение технологии XSLT при разработке сайтов на UMI.CMS

Юмисофт 2010

Оглавление

От автора

В этой книге рассматриваются основы использования языка XSLT при разработке сайтов на примере XSLT-шаблонизатора UMI.CMS. «Основы» — в данном случае это не только краткие указания на то, с чего следует начинать, но и необходимая и достаточная информация, для того, чтобы можно было двигаться дальше и развиваться самостоятельно.

Важно понимать, что эта книга не справочник и не полное руководство по XSLT — такие книги существуют давно, и в них можно найти исчерпывающую информацию по всем тонкостями и нюансам технологии. Однако опыт показывает, что у большинства разработчиков может не быть времени на чтение справочников и что многим бы хотелось сократить время знакомства, получив, тем не менее, достаточно знаний, чтобы решать практические задачи. Именно поэтому в этой книге большая часть материала построена исключительно на практических примерах, имеющих непосредственное отношение к разработке сайтов.

Кроме того, следует иметь в виду, что знание всех тонкостей и нюансов XSLT не понадобится при разработке сайтов на UMI.CMS – однако, в случае необходимости, знания из этой книги помогут в дальнейшем изучить их самостоятельно.

Важно также понимать, что эта книга не справочник по разработке сайтов на UMI.CMS — для этого есть документация, примеры в документации и отдельный сайт wiki, посвященный этой системе. Здесь указаны главные направления и подходы, которые помогут правильно использовать XSLT-шаблонизатор UMI.CMS и позволят избежать целого ряда возможных ошибок в будущем.

Таким образом, главной целью этой книги было коротко и лаконично (чтение не должно занять более одного вечера) на практических примерах показать, как можно использовать XSLT при разработке сайтов, почему это несложно, и что это не обязательно требует серьезных теоретических знаний.

I. Модель данных UMI.CMS

Любой разработчик сайтов рано или поздно сталкивается с необходимостью разобраться с тем, что называется «моделью данных» системы, с которой он работает. И лучше это произойдет раньше, чем позже, так как позволит сэкономить прежде всего время разработчика и его бесценные нервы.

Основные сведения

Сейчас мы рассмотрим лишь основные положения модели данных UMI.CMS, за деталями рекомендуется обратиться в документацию разработчика сайтов[1].

1 **Все данные в UMI.CMS хранятся в виде объектов и больше никаких сущностей в модели данных нет.**

Иными словами: страницы контента, баннеры, товары, скидки, пользователи — все это объекты системы. Набор характеристик или свойств объекта определяется «полями», которые группируются в «группы полей».

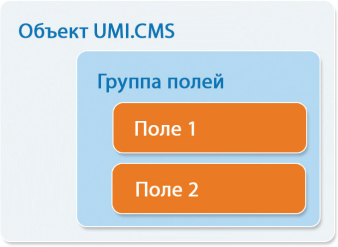

Рисунок 1. Объект UMI.CMS, группа полей и поля

[1] http://help-dev.umi-cms.ru/

Все объекты создаются по шаблонам, определенным в системе. Эти шаблоны называются «типами данных» и именно они определяют характеристики объектов.

Разработчикам сайта предоставлен мощный инструмент — модуль «Шаблоны данных». Этот модуль позволяет добавлять и модифицировать шаблоны для создания объектов, изменяя набор полей или сами поля в типах данных. Таким образом, можно быстро, при помощи удобного административного интерфейса, настраивать объекты системы под задачи конкретного проекта.

Замечание: Одним из следствий этого подхода является то, что база данных UMI.CMS оптимизирована для хранения объектов и использует технологию ORM[1]. Поэтому предполагается, что у разработчика сайта не должно возникнуть необходимости работать с базой данных напрямую.

Страницы в UMI.CMS — это объекты, которые являются документами в структуре сайта, и таким образом связаны с иерархией в дереве сайта.

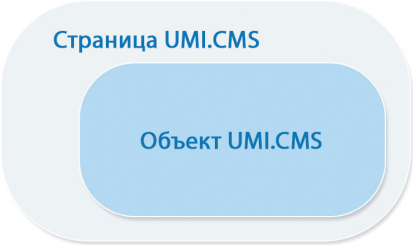

Рисунок 2. Страница UMI.CMS — объект с дополнительными свойствами

[1] http://ru.wikipedia.org/wiki/ORM

В связи с этим у страницы появляется набор дополнительных свойств: таких как URL, положение в иерархии сайта, и другие. Полный список мы здесь рассматривать не будем — часть мы обсудим чуть позже, оставшуюся часть можно посмотреть в документации разработчика UMI.CMS.

Для работы с данными системы используется функционал UMI.CMS, реализованный при помощи API и макросов, а для вывода данных в необходимом виде — используются шаблонизаторы.

Таким образом, разработчику сайтов на UMI.CMS предоставлен еще один мощный инструмент — шаблонизатор, позволяющий отделить логику от представления и воспользоваться в полной мере преимуществами этого подхода.

Далее мы подробно рассмотрим принципы разработки сайтов при помощи XSLT-шаблонизатора UMI.CMS.

Сайт как набор XML-сервисов

Подход к разработке сайта на UMI.CMS с использованием XSLT-шаблонизатора, скорее всего, отличается от привычного. Прежде чем двигаться дальше, давайте остановимся на следующей мысли:

При разработке сайта с использованием XSLT-шаблонизатора мы работаем с данными UMI.CMS в формате XML

Например:

- Вывод страниц в браузер — обработка XML-данных.
- Вывод меню сайта — обработка XML-данных.
- Вывод товаров каталога — обработка XML-данных.

- Вывод фотогалереи — обработка XML-данных.
- Вывод формы авторизации пользователя — обработка XML-данных.
- И так далее.

Иными словами, можно и следует воспринимать сайт, работающий на UMI.CMS — как набор различных XML-сервисов.

Рисунок 3. Подключение различных XML-данных для вывода страницы

Чтобы собрать все необходимые данные вместе и показать их в браузере именно так, как мы хотим — нам и нужен шаблонизатор. Таким образом, основная и единственная задача шаблонизатора — это вывод данных.

Язык XSLT выбран не случайно — на сегодняшний день это идеальный инструмент для работы с XML-данными.

Рассмотрим для примера, как выглядят объекты и страницы UMI.CMS в виде XML. Заодно убедимся в том, насколько это удобно и наглядно.

Объекты в UMI.CMS: объект в виде XML

Примеры объектов в виде XML: пользователь и баннер

Для того чтобы посмотреть как выглядит любой объект в UMI.CMS в виде XML возьмем, к примеру, две разных сущности: пользователя и баннер.

nb **Стоит запомнить:** UMI.CMS предоставляет возможность просмотреть содержимое любого существующего объекта системы в виде XML непосредственно в окне браузера.

Для того чтобы вывести объект в виде XML, можно набрать в адресной строке http://адрес_сайта/uobject/id_объекта, где вместо адрес_сайта, следует вписать адрес вашего сайта на UMI.CMS, а вместо id_объекта — реальный id объекта системы.

На самом деле — это вызов по одному из протоколов UMI.CMS, их мы обсудим далее. Этот вызов может быть запрещен настройками системы (см. главу «Общие замечания по протоколам UMI.CMS»).

Узнать id можно на странице редактирования объекта в административном интерфейсе. Id будет виден в URL, например:
http://адрес_сайта/admin/banners/edit/27531/

```
<udata generation-time="0.000062">
   <object id="14" name="sv" type-id="4">
      <properties>
         <group id="16" name="idetntify_data">
            <title>Идентификационные данные</title>
            <property id="45" name="login" type="string">
               <title>Логин</title>
               <value>sv</value>
            </property>
            +<property id="12" name="e-mail" type="string">
            +<property id="53" name="groups" type="relation"
            multiple="multiple">
            +<property id="8735" name="is_activated" type="boolean">
            +<property id="8967" name="last_request_time" type="int">
         </group>
         +<group id="5" name="short_info">
      </properties>
   </object>
</udata>
```

Пример 1. Пользователь с id = 14 — супервайзер, выбранный при установке системы

```
<udata generation-time="0.001826">
   <object id="29764" name="Баннер с картинкой" type-id="672" ownerId="14">
      <properties>
         +<group id="2939" name="common">
         <group id="2945" name="redirect_props">
            <title>Параметры перехода</title>
            <property id="8800" name="url" type="string">
               <title>Url страницы</title>
               <value>http://example.ru</value>
            </property>
            +<property id="8801" name="open_in_new_window" type="boolean">
         </group>
         +<group id="2942" name="view_pages">
         +<group id="2943" name="banner_custom_props">
      </properties>
   </object>
</udata>
```

Пример 2. Баннер с id = 29764

Мы говорили ранее о том, что свойства объекта (поля) сгруппированы в группы полей. В этих примерах можно видеть элементы <group>, внутри которых находятся один или более тегов <property>. Это и есть группы полей, и отдельные поля. Значение полей заключено в элементе <value>.

Таким образом, мы имеем перед глазами наглядное представление любого объекта системы — мы можем не только представить структуру данных, но и узнать значения конкретных свойств.

Важно: в этих примерах можно видеть, что кроме значений и структуры, мы можем получить информацию об идентификаторах, названиях и типах данных для полей объекта — эта информация содержится в атрибутах тегов <property>. Как мы можем ей воспользоваться — мы поговорим далее.

Страницы в UMI.CMS: страницы в виде XML

Пример страницы в виде XML: страница контента и страница объекта каталога

Для того чтобы посмотреть как выглядит любая страница в UMI.CMS в виде XML возьмем, к примеру, две разных страницы: страницу контента и страницу объекта каталога.

Стоит запомнить: UMI.CMS предоставляет возможность просмотреть содержимое любой существующей страницы в системе в виде XML непосредственно в окне браузера.

Для того, чтобы вывести страницу в виде XML следует набрать в адресной строке http://адрес_сайта/upage/id_страницы, где вместо адрес_сайта, следует вписать адрес вашего сайта на UMI.CMS, а вместо id_страницы — реальный id страницы системы.

На самом деле — это вызов по одному из протоколов UMI.CMS, их мы обсудим далее. Этот вызов может быть запрещен настройками системы (см. главу «Общие замечания по протоколам UMI.CMS»).

Узнать id можно, зайдя на страницу редактирования страницы сайта в административном интерфейсе. Id будет виден в URL, например: http://адрес_сайта/admin/content/edit/70/

Важно: В связи с тем, что страницы, по сути являются элементами иерархии в структуре сайта, для них используются свои собственные числовые id, не совпадающие с id объектов.

```
<udata generation-time="0.002269">
   <page id="70" parentId="0" link="/help/" is-active="1"
   object-id="27660" type-id="10" update-time="1274273454" alt-name="help">
      <basetype id="2" module="content">Страницы контента</basetype>
      <name>Помощь</name>
      <properties>
         <group id="22" name="common">
            <title>Основные параметры</title>
            +<property id="22" name="title" type="string">
            <property id="23" name="h1" type="string">
               <title>Поле H1</title>
               <value>Помощь</value>
            </property>
            +<property id="26" name="content" type="wysiwyg">
         </group>
         +<group id="24" name="more_params">
      </properties>
   </page>
</udata>
```

Пример 3. Страница контента с id = 70

```
<udata generation-time="0.002384">
   <page id="7" parentId="4" link="/shop/dvd_tehnika/toshiba_srq660/"
   is-active="1" object-id="27596" type-id="828" update-time="1274273439"
   alt-name="toshiba_srq660">
      <basetype id="6" module="catalog"
      method="object">Объекты каталога</basetype>
      <name>Toshiba SR-Q660</name>
      <properties>
         +<group id="3385" name="common">
         <group id="3389" name="cenovye_svojstva">
            <title>Ценовые свойства</title>
            <property id="8918" name="price" type="price">
               <title>Цена</title>
               <value xlink:href="udata://emarket/price/27596">310</value>
            </property>
         </group>
         +<group id="3394" name="opisanie_tovara">
         +<group id="3396" name="item_properties">
      </properties>
   </page>
</udata>
```

Пример 4. Страница объекта каталога с id = 7

Как и в случае объектов, мы видим, что свойства страницы (поля) сгруппированы в группы полей: можно видеть элементы <group>, внутри которых находятся один или более тегов <property>. Значение полей у страниц также заключено в элементе <value>.

Таким образом, мы также имеем перед глазами наглядное представление любой страницы системы: структуру ее данных, и значения конкретных свойств.

Важно: точно так же как и у объектов, нам доступны атрибуты полей. Кроме того, можно видеть, что в атрибутах тега <page> содержится также полезная дополнительная информация: URL страницы, id родительской страницы, и так далее.

Когда мы обсуждали модель данных, мы упоминали, что страницы — это тоже объекты. Это утверждение легко проверить. В атрибуте **object-id** тега <page> содержится id, по которому эту страницу можно получить как объект.

Например, страница новости может быть представлена в следующем виде:

```
<udata generation-time="0.002161">
   <object id="27656" name="Только семь дней" type-id="23" ownerId="14">
      <properties>
         <group id="75" name="common">
            <title>Основные параметры</title>
            +<property id="23" name="h1" type="string">
            <property id="71" name="anons" type="wysiwyg">
               <title>Анонс</title>
               <value>
                  <p>Только в течение семи дней вы можете получить
                  персональную скидку на новинки нашего магазина!</p>
               </value>
            </property>
            +<property id="26" name="content" type="wysiwyg">
         </group>
      </properties>
   </object>
</udata>
```

Пример 5. Страница новости в виде объекта (http://адрес_сайта/uobject/27656)

Однако как можно видеть, в этом случае получены лишь поля и группы полей объекта. Та информация, которая характеризует этот объект как страницу — информация о положении в иерархии, URL страницы, информация о модуле и методе и другое — все это здесь недоступно.

Резюме:

- В рассмотренных примерах можно видеть, что использование формата XML гарантирует разработчику единообразие и естественность представления данных.
- Данные об объектах и страницах системы можно получить в исключительно наглядном и структурированном виде, в любой момент разработки, запросив XML-представление непосредственно в окне браузера.
- Из этого следует, что можно реже обращаться в документацию, так как всегда доступны примеры реальных данных, которые используются при разработке сайта.

II. XSLT-шаблонизатор UMI.CMS

В предыдущей части мы рассмотрели то, как выглядят объекты и их частный случай — страницы в UMI.CMS. Теперь, прежде чем приступить к разговору о XSLT-шаблонизаторе, нам необходимо познакомиться еще с одним понятием: форматом UMIData.

Вывод страницы в браузере: формат UMIData

Данные в формате UMIData — это те XML-данные, которые система формирует в ответ на запрос любой страницы сайта. Эти данные обрабатываются XSLT-шаблонизатором, после чего результат выводится в окно браузера.

Это можно выразить одной простой формулой:

Запрос из браузера ⟶ UMIData + XSLT-шаблон страницы ⟶ HTML код в браузере

Или еще проще диаграммой:

Рисунок 4. Формирование HTML-кода XSLT-шаблонизатором

Стоит запомнить: Что именно обрабатывается шаблонизатором при запросе той или иной страницы (страницу в формате UMIData) можно в любой момент посмотреть, добавив к адресу страницы «.xml».

Например: http://адрес_сайта/.xml или http://адрес_сайта/about/.xml

Рассмотрим в качестве примера ответ UMIData на запрос страницы контента:

```
<result module="content" method="content" domain="example.ru" lang="ru"
header="Контактная информация" title="UMI.CMS - Контактная информация"
request-uri="/about/contacts/.xml" pageId="71">
   <meta>
      <keywords>umi CMS демо DEMO сайт система управление</keywords>
      <description>Контактная информация</description>
   </meta>
   <user id="14" type="sv" status="auth" login="sv"
   xlink:href="uobject://14"/>
   <parents>
      <page id="68" parentId="0" link="/about/" is-active="1"
      object-id="27658" type-id="10" update-time="1274273454"
      alt-name="about" xlink:href="upage://68">
         <basetype id="2" module="content">Страницы контента</basetype>
         <name>О магазине</name>
      </page>
   </parents>
   <page id="71" parentId="68" link="/about/contacts/" is-active="1"
   object-id="27661" type-id="826" update-time="1276599076"
   alt-name="contacts">
      <basetype id="2" module="content">Страницы контента</basetype>
      <name>Контактная информация</name>
      <properties>
         +<group id="3366" name="common">
         +<group id="3368" name="more_params">
         +<group id="3372" name="site_info">
      </properties>
   </page>
</result>
```

Пример 6. Ответ UMIData для страницы контента

Можно видеть, что атрибуты тега <result> содержат важную информацию, такую как: модуль и метод, вернувшие эту страницу, домен, язык, заголовок,

title, GET-параметры и идентификатор страницы.

Далее мы можем видеть, что внутри тега <result> находятся четыре элемента, каждый со своим содержимым:

- <meta> — описывает мета-информацию страницы.
- <user> — описывает пользователя, запросившего страницу.
- <parents> — описывает родительские страницы, если они есть.
- <page> — описывает саму страницу; мы уже знакомились с этой структурой в предыдущей главе.

Подробное описание формата UMIData можно посмотреть в документации разработчика сайта. Сейчас главное понять, что у нас есть доступ к этой информации сразу после запроса страницы из браузера. Как мы покажем далее, любую информацию, содержащуюся в UMIData, мы можем вывести в шаблонах.

Резюме:

- UMIData — это данные в формате XML. Следовательно, все обсуждавшиеся ранее плюсы этого формата, актуальны и здесь: единообразие, наглядность и структурированность.
- UMIData — это данные, которые обрабатывает XSLT-шаблонизатор. Структуру этих данных можно получить в любой момент разработки, запросив представление непосредственно в окне браузера — следовательно, мы всегда точно знаем, к чему именно применяются шаблоны.
- Сразу после запроса страницы доступна системная информация об этой странице в теге <result>
- Сразу после запроса страницы доступна информация о пользователе, открывшем страницу (в теге <user>).
- Сразу после запроса страницы доступна информация о положении в иерархии (в теге <parents>).
- Сразу после запроса страницы доступна информация о полях запрошенной страницы (в теге <page>).

Основные сведения об XSLT

В первой части мы говорили о том, что можно воспринимать сайт на UMI.CMS, как набор XML-сервисов. Кроме того, мы постарались показать удобство и наглядность представления данных в формате XML.

Следующие несколько глав будут посвящены объяснению основных принципов языка XSLT, на примерах практического использования его в качестве языка шаблонизатора UMI.CMS.

Мы не будем рассматривать здесь все детали и тонкости языка, а сконцентрируемся на необходимых знаниях, которых будет достаточно, чтобы начать разрабатывать сайты на UMI.CMS с использованием технологии XSLT.

Для того чтобы двигаться дальше, запомним это определение:

> XSLT — это язык преобразований одних XML-документов в другие XML-документы

В наших задачах мы будем преобразовывать XML-данные в HTML-код.

Небольшое отступление

Существует мнение, что технология XSLT — это сложно, медленно, неудобно, слишком абстрактно и оторвано от реального процесса разработки сайтов.

Причины, по которым сложились подобные убеждения, можно коротко свести к следующим:

- **Документация из прошлого века.** Спецификация и часть литературы написаны еще в конце 1990х годов, и материал излагается либо с точки зрения абстрактного подхода, либо в примерах, имеющих слабое отношение к современным реалиям WWW.

- **Опыт известных технологий переносится по привычке, без попыток разобраться в концепции технологии XSLT.** Чтобы сэкономить время изучения, разработчики «цепляются» за знакомые конструкции, так как они напоминают им аналогичные конструкции в других ранее изученных языках.
- **Попытки решать задачи, для которых язык XSLT не предназначен.** Помните — этот язык предназначен для *преобразования* XML-данных. Попытки использовать его в каких-либо других целях, приводят к закономерным «сложностям», «медленности» и «неудобствам».
- **Неправильное использование и примеры в сети, построенные на этом.** Этот пункт является прямым следствием предыдущих. В поисках примеров практического применения разработчики зачастую натыкаются на «примеры», которые не только не помогают, а лишь закрепляют иллюзию «сложности» и «неудобности».

Вывод же всего один: любой инструмент надо использовать правильно.

В этом случае вы убедитесь, что XSLT не только может использоваться в разработке сайтов, но и оказывается чрезвычайно удобен для этого. Что мы и продемонстрируем на примерах, объясняющих принципы разработки сайтов на UMI.CMS с использованием XSLT-шаблонизатора.

Таблица стилей XSLT — это также XML

Мы будем рассматривать основы языка XSLT в виде положений. Эти положения мы будем формулировать по мере изучения, поэтапно, в тот момент, когда они нам понадобятся.

1 **Документ таблицы стилей, по которым производится преобразование исходного документа XML — это также документ XML.**

Важным следствием этого положения является то, что, как и всякий документ XML, документ с таблицей стилей должен быть то, что называется **“well-formed”** [1].

Поэтому, когда мы используем XSLT в качестве языка шаблонизатора, нам следует иметь в виду следующее:

- Все теги должны быть закрыты и правильно вложены.
- Все атрибуты должны быть заключены в кавычки.
- Регистр имеет значение.
- Весь XML-документ должен находиться внутри одного элемента.
- Изначально определены лишь некоторые сущности: & < > ' "[2]

Кроме того, еще одним следствием является то, что документы таблицы стилей, также как и другие XML-документы могут быть преобразованы при помощи других таблиц стилей.

Замечание: элемент XML — это открывающий тег элемента, закрывающий тег элемента и все содержимое между ними

Давайте посмотрим на простейший файл таблицы стилей изнутри:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" />

   <xsl:template match="/">

      <!-- код шаблона -->

   </xsl:template>

</xsl:stylesheet>
```

Пример 7. Простейший файл таблицы стилей XSLT

[1] Можно посмотреть подробнее: http://www.w3.org/TR/xml/#sec-well-formed

[2] Эти сущности следует использовать вместо &, <, >, ‘, “ соответственно

В этом файле мы видим:

- В первой строке объявление, говорящее о том, что это XML-документ.
- Элемент <xsl:stylesheet> говорит о том, что это таблица стилей XSLT; это тот элемент, в котором содержится вся таблица стилей.
- Элемент <xsl:output> указывает, в каком виде мы хотим получить итоговый документ.
- Элемент <xsl:template> определяет единственный в этой таблице стилей шаблон; о шаблонах мы поговорим чуть позже.

Подробное описание атрибутов у первых двух элементов, в случае необходимости, вы можете посмотреть самостоятельно[1].

Резюме:

При верстке шаблонов на XSLT следует быть внимательнее и следить за «правильностью» кода, в том числе и HTML

[1] http://www.w3.org/TR/xslt

Представление XML-документов в процессе обработки

XSLT-шаблонизатор работает с XML-документами в виде «деревьев» — как с входящим документом, так и с таблицей стилей, и с итоговым документом. Это абстрактное представление, скорее всего, в полной мере вам не понадобится при разработке сайта, но мы рассмотрим его сейчас кратко, для понимания некоторых принципов, связанных с шаблонами в таблице стилей.

Рассмотрим документ UMIData в виде дерева:

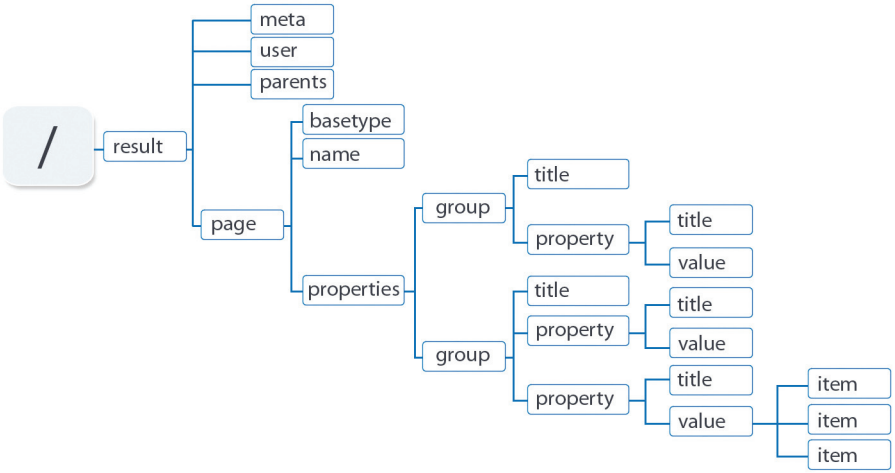

Рисунок 5. Документ UMIData в виде дерева

Как вы можете видеть, все элементы находятся внутри элемента result, причем сохранен порядок и вложенность. Теперь становится понятно, зачем нужно строго следить за правильно вложенными и закрытыми тегами — иначе это дерево не будет построено.

Слева на схеме обозначен "/" так называемый «корень дерева» или «корневой узел», который содержит весь документ, в том числе и корневой элемент result.

Помните, чуть выше мы рассматривали пример простейшего файла таблицы стилей? Мы говорили о том, что в этом документе элемент <xsl:template> определяет шаблон. На самом деле атрибут **match="/"** у этого элемента говорит о том, что этот шаблон необходимо применить к корню входящего XML-документа (то есть «ко всему документу»). Почему это так, мы в деталях рассмотрим позже, пока просто запомним это.

Теперь давайте разберемся, как можно ориентироваться внутри XML-документа. Посмотрим еще раз на упоминавшийся выше пример:

```
<result module="content" method="content" domain="example.ru" lang="ru"
header="Контактная информация" title="UMI.CMS - Контактная информация"
request-uri="/about/contacts/.xml" pageId="71">
   <meta>
      <keywords>umi CMS демо DEMO сайт система управление</keywords>
      <description>Контактная информация</description>
   </meta>
   <user id="14" type="sv" status="auth" login="sv"
   xlink:href="uobject://14"/>
   <parents>
      <page id="68" parentId="0" link="/about/" is-active="1"
      object-id="27658" type-id="10" update-time="1274273454"
      alt-name="about" xlink:href="upage://68">
         <basetype id="2" module="content">Страницы контента</basetype>
         <name>О магазине</name>
      </page>
   </parents>
   <page id="71" parentId="68" link="/about/contacts/" is-active="1"
   object-id="27661" type-id="826" update-time="1276599076"
   alt-name="contacts">
      <basetype id="2" module="content">Страницы контента</basetype>
      <name>Контактная информация</name>
      <properties>
         +<group id="3366" name="common">
         +<group id="3368" name="more_params">
         +<group id="3372" name="site_info">
      </properties>
   </page>
</result>
```

Пример 8. XML-документ в формате UMIData

Адресация внутри XML-документа — язык запросов XPath

Представим себе XML-документ UMIData в виде подобия файловой системы с вложенными папками и подпапками:

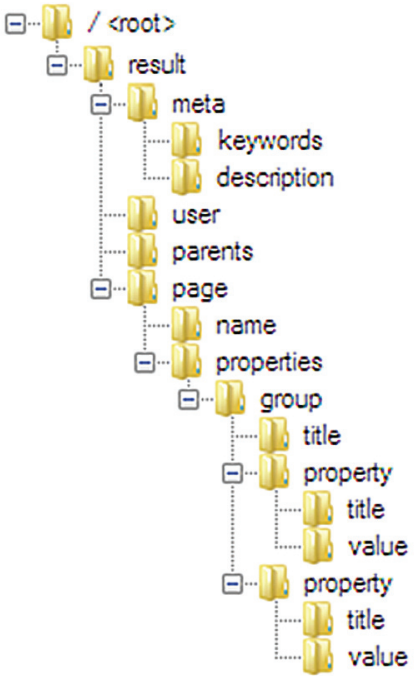

Рисунок 6. Документ UMIData в виде подобия файловой системы

В такой приближенной интерпретации можно считать, что элемент XML-документа — это «папка» со всем ее содержимым.

Замечание: в подобном виде (с раскрывающимися узлами) мы легко можем рассматривать данные UMIData (как впрочем, и остальные XML-данные), если воспользуемся браузером Mozilla Firefox. В браузере нам будут видны также атрибуты элементов и их значения.

В таком представлении можно легко указать нужный элемент, записав путь до него:

- /result/meta/keywords — укажет на элемент keywords.
- /result/page/properties/group — укажет на элемент group.

Для того чтобы указать атрибут необходимо написать символ @:

- /result/@title и /result/@pageId — title для страницы и ее id соответственно (атрибуты тега result).
- /result/page/@link — URL открытой страницы (атрибут тега page).
- /result/user/@login — логин, открывшего страницу пользователя (атрибут тега user).

Вопрос: а что делать, если мы хотим указать какой-то конкретный элемент? Например, свойство, содержащее значение заголовка? (для страниц UMI.CMS – это элемент property, у которого атрибут name равен "h1")

В таком случае, мы можем указать условие при помощи квадратных скобок:

- /result/page/properties/group/property[@name = 'h1'] — укажет на элемент, удовлетворяющий условию и находящийся по указанному адресу.
- /result/page/properties/group[@name = 'common']/title — укажет на заголовок группы полей, удовлетворяющей указанному условию.

Важно: следует иметь в виду, что если указанным условиям удовлетворяет больше 1 элемента, запрос XPath вернет множество элементов, удовлетворяющих условиям запроса. Например:

/result/page/properties/group[property] — вернет все элементы group, внутри которых есть хотя бы один тег property.

Из этого примера видно также, что мы можем указать в квадратных скобках не только равенство, но и условие наличия какого-либо элемента.

Понимание этих примеров — запись путей адресации и условий языка XPath, вплотную приближает нас к понятию «образцов» (паттернов), которыми мы будем дальше активно пользоваться, когда займемся вопросами применения шаблонов из таблиц стилей.

Резюме:

- Адресация XPath позволяет получить доступ к любой информации, содержащейся внутри XML-документа.
- Структуру XML-файла UMIData всегда можно посмотреть в окне браузера — поэтому написание путей и условий не представляет сложности.
- Адресация XPath позволяет получить доступ не только к отдельному элементу, но и группе элементов, подходящих под заданное условие (образец или паттерн).

Убедимся, что шаблон **<xsl:template match="/">** действительно применяется ко всему исходному документу и выведем значения из исходного документа.

Вывод значений: <xsl:value-of> и { }

Теперь, когда мы знаем, как можно обратиться к произвольному элементу или его атрибуту, нам не составит труда вывести значение этого элемента.

Подключаем шаблон страницы UMI.CMS

Проделаем несколько подготовительных операций:

- Немного видоизменим приведенную ранее таблицу стилей:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" />

      <xsl:template match="/">
         <html>
            <head>
               <title>Тайтл</title>
            </head>
            <body>
               <div id="content">
                     <h1>Заголовок</h1>

                        ... тестируем шаблон ...

               </div>
            </body>
         </html>
      </xsl:template>

</xsl:stylesheet>
```

Пример 9. Тестовая таблица стилей

- Создадим файл с этой таблицей стилей в папке /xsltTpls/, назовем его "default.xsl". Этот файл должен быть в кодировке UTF-8.
- Зайдем в административную панель, в настройки модуля «Структура» и в нижнем поле напишем название для шаблона страниц, допустим «XSLT-шаблон», а справа напишем название созданного только что файла — "default.xsl".

- Поставим опцию «основной» и нажмем «Сохранить».

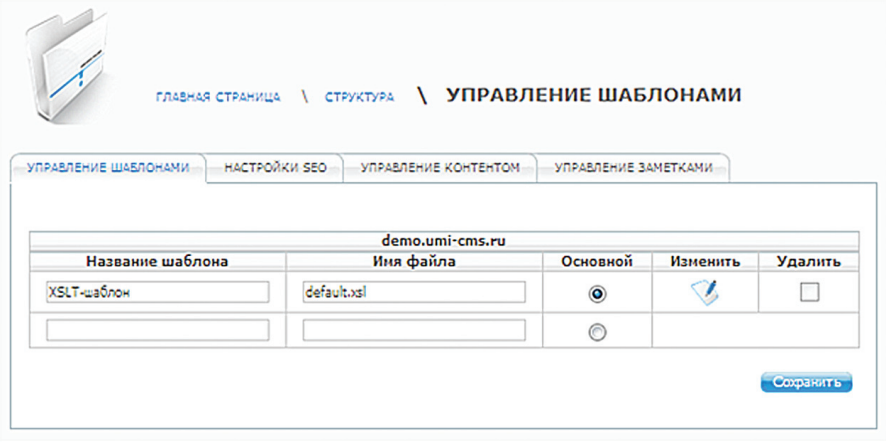

Рисунок 7. Подключение XSLT-шаблона

Замечание: вы можете использовать UMI.CMS localpack, или любой вам доступный тестовый сайт на UMI.CMS, к которому у вас есть доступ по FTP и в административную панель.

Теперь этот файл таблицы стилей может применяться к любой странице сайта. Вспомним формулу, которую мы приводили, когда рассматривали формат UMIData:

Запрос из браузера ⟶ UMIData + XSLT-шаблон страницы ⟶ HTML код в браузере

Если все сделано правильно, теперь вместо страниц сайта, выводящихся по этому шаблону, мы увидим текст «Заголовок» и «… тестируем шаблон …». Это означает, что текст из шаблона выводится.

Итак, мы хотим удостовериться, что, применяя **<xsl:template match="/">** ко всему документу UMIData, у нас есть доступ к элементам этого документа.

Инструкция <xsl:value-of>

Познакомимся с новой для нас инструкцией XSLT:

- **Вывод значений элементов или атрибутов в шаблоне осуществляется при помощи инструкции <xsl:value-of select="путь_до_элемента">**

Мы хотели вывести следующее:

- TITLE – получить его можно, указав /result/@title
- Заголовок – получить его можно, указав /result/@header
- Контент – получить его можно, указав /result/page/properties/group/property[@name = 'content']/value

Воспользуемся инструкцией <xsl:value-of> в нашей таблице стилей и укажем пути до нужных элементов. Отредактируем файл default.xsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" />

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="/result/@title"/></title>
         </head>
         <body>
            <div id="content">
               <h1><xsl:value-of select="/result/@header"/></h1>

                  <xsl:value-of
                  select="/result/page/properties/group/property[@name
                  = 'content']/value"/>
            </div>
         </body>
      </html>
   </xsl:template>

</xsl:stylesheet>
```

Пример 10. Вывод значений при помощи <xsl:value-of>

Теперь на страницах сайта будут выводиться нужные нам значения. Однако, для того, чтобы содержимое поля «контент» выводилось без экранирования тегов, необходимо добавить еще один атрибут, записав инструкцию так:

<xsl:value-of select="result/page/properties/group/property[@name = ‘content’]/value" disable-output-escaping="yes"/>

Чтобы не записывать такой длинный путь до элемента, можно воспользоваться оператором ‘//’, который позволяет найти среди всего дерева на любом уровне вложенности элемент, удовлетворяющий условию.

Предыдущую инструкцию тогда можно записать так:

<xsl:value-of select="//property[@name = ‘content’]/value" disable-output-escaping="yes"/>

Замечание: следует иметь в виду, что поиск по дереву может быть требователен к ресурсам, поэтому указание полного пути до элемента, в целом, предпочтительнее.

Фигурные скобки в атрибутах: { }

Часто может возникнуть необходимость вывести какие-либо значения внутри атрибута тега, например:

<meta name="description" content=""/>
<meta name="keywords" content=""/>

В таком случае необходимо воспользоваться записью через фигурные скобки, указав таким же образом путь до элемента:

<meta name="description" content=**"{result/meta/description}"**/>
<meta name="keywords" content=**"{result/meta/keywords}"**/>

Резюме:

- Использование XSLT и адресации XPath позволяет выводить действительно любые данные, относящиеся к исходному XML-файлу (в нашем случае — к открытой странице) при помощи единственной инструкции <xsl:value-of>
- Использование XSLT и адресации XPath позволяет выводить данные также внутри атрибутов, при помощи фигурных скобок.

Образцы и шаблоны: использование <xsl:apply-templates> + <xsl:template>

Теперь, когда мы уже знаем возможности языка запросов XPath, мы можем в полной мере задействовать его в процессе преобразования исходного XML-документа. Для этого познакомимся со вторым положением языка XSLT.

Преобразование документа осуществляется путем связывания образцов с шаблонами

Мы уже говорили, что шаблоны в таблице стилей создаются при помощи инструкции <xsl:template>.

Важно: договоримся с этого момента, что «шаблонами» мы будем называть именно этот элемент — от открывающего тега <xsl:template>, до закрывающего тега </xsl:template>. Файлы, которые мы назначаем «шаблонами» страниц в настройках модуля «Структура» мы будем называть «файлами таблиц стилей».

Мы видели также, что у шаблона есть атрибут **match**. В случае шаблона для корня документа, он был равен " / " — это и был тот самый «образец» для корневого узла документа.

Мы говорили также, что можно составить запрос XPath так, что он будет указывать не на один элемент в XML-файле, а на множество элементов, удовлетворяющих условию и находящихся по указанному пути. В общем смысле — такой запрос и есть «образец» или «паттерн» для этих элементов.

Итак: *образец или паттерн* — **это выражение XPath, идентифицирующее элемент или группу элементов в XML-документе**[1].

Давайте посмотрим теперь на то, как работает принцип связывания образцов и шаблонов. Для этого познакомимся еще с одной инструкцией XSLT.

Инструкция <xsl:apply-templates>

Отредактируем код нашей таблицы стилей и допишем еще одни шаблон при помощи <xsl:template>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>
   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>
</xsl:stylesheet>
```

Пример 11. Использование <xsl:apply-templates> и <xsl:template>

[1] Те, кто знаком с понятием селекторов в CSS, совершенно справедливо могут отметить очевидные аналогии.

Инструкция <xsl:template> создает шаблон. Условие match="result" связывает этот шаблон с образцом, к которому он должен быть применен.

Запись **<xsl:apply-templates select="result"/>** означает следующее: **выбрать элемент result и все его содержимое и применить к нему шаблоны.** Получившийся результат, как и в случае с <xsl:value-of>, будет выведен вместо инструкции.

Доступные шаблоны — это шаблоны, созданные инструкцией <xsl:template>. XSLT-процессор будет искать среди всех шаблонов, расположенных в таблице стилей, сравнивая выбранный элемент с образцами шаблонов.

Если мы посмотрим на наш файл с таблицей стилей мы можем заметить, что там находится 2 элемента <xsl:template>. Однако подходит из них только один. Почему?

- У первого связь с образцом задана при помощи match=" / " — так записывается соответствие корню документа.
- У второго связь с образцом задана при помощи match="result" — и это имя того элемента, который мы выбрали в <xsl:apply-templates>

В <xsl:apply-templates> мы указали select="result" и образец для сравнения задан во втором шаблоне как match="result" – поэтому будет выбран именно второй шаблон.

Важно: заметьте также, что поскольку мы выбрали элемент result, то внутри применяемого шаблона адресация ведется уже от элемента result (в аналогии с файловой системой это означало бы, что мы зашли в «папку» result). Поэтому мы выводим 2 артибута тега result (модуль и метод), записав @module и @method. Как говорится, у нас изменился контекст.

Если мы теперь обновим страницы нашего сайта, то мы увидим, что под заголовком будет выведено 2 строки, которые сообщают, каким модулем и каким методом этого модуля выводится данная страница.

Казалось бы, ничего не изменилось: мы прекрасно могли бы вывести те же атрибуты из первого шаблона с match="/ "?

Чтобы понять, что это нам дает, давайте допишем еще один шаблон:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
form">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>

            <h1><xsl:value-of select="result/@header"/></h1>

            <xsl:apply-templates select="result"/>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="result[@module = 'content'][@method = 'content']">
        <p>это - страница контента</p>
   </xsl:template>

</xsl:stylesheet>
```

Пример 12. Использование уточняющих условий в образцах

Теперь, если мы откроем любую страницу контента, мы увидим надпись «это — страница контента». Тогда как на остальных страницах сайта по-прежнему будут выводиться модуль и метод UMI.CMS, выводящие эти страницы.

Давайте разберемся почему. Рассмотрим, что происходит при запросе страницы из браузера:

- Когда мы запрашиваем в браузере страницу контента, в ответ система формирует XML-документ в формате UMIData. Этот документ будет содержать элемент result с атрибутами module равным ‘content’ и method, равным ‘content’. В этом легко убедиться, дописав ".xml" к открытым страницам контента.
- Сначала, шаблонизатор будет искать шаблон для корня документа — это шаблон с образцом match="/ ".
- Далее в шаблоне <xsl:template match="/"> стоит инструкция <xsl:apply-templates>, которая выберет элемент result с его содержимым, и укажет, что его нужно попытаться обработать по доступным шаблонам.

Теперь шаблонов 3:

- У первого связь с образцом задана при помощи match="/ "
- У второго связь с образцом задана при помощи match="result"
- У третьего связь с образцом задана при помощи match="result [@module = ‘content’][@method = ‘content’]"

Вопрос: почему будет выбран третий шаблон, а не второй?

Потому что у него образец задан *более точно*: в тех случаях, когда элемент result содержит атрибуты с такими значениями, у этого шаблона будет *более высокий приоритет*.

Как видите, мы используем те же правила XPath для создания образцов в шаблонах, что и при адресации внутри XML-документа, когда мы выводили значения.

Различаем страницы сайта

Давайте попробуем теперь добавить еще несколько шаблонов, дописав еще несколько элементов <xsl:template>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>
   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
         <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>
   <xsl:template match="result[@module = 'content'][@method = 'content']">
      <p>это - страница контента</p>
   </xsl:template>
   <xsl:template match="result[@module = 'news'][@method = 'rubric']">
      <p>это - страница ленты новостей</p>
   </xsl:template>
   <xsl:template match="result[@module = 'news'][@method = 'item']">
      <p>это - страница отдельной новости</p>
   </xsl:template>
   <xsl:template match="result[page/@is-default = '1']">
      <p>это – главная страница сайта</p>
   </xsl:template>
   <xsl:template match="result[not(page)]">
      <p>это – системная страница</p>
   </xsl:template>
   <xsl:template match="result[//property[@name = 'my_property']]">
      <p>это – страница со свойством 'my_property'</p>
   </xsl:template>
</xsl:stylesheet>
```

Пример 13. Использование уточняющих условий в образцах

Видите, мы по-прежнему используем все ту же запись адресации в XML-документе, но немного изменили условия:

- Страницы ленты новостей и страницы полного текста новости мы различили точно так же как и страницы контента — по атрибутам, описывающим модуль и метод.
- Условие для главной страницы match="result[page/@is-default = ‘1’]" говорит о том, что этот шаблон надо применять к элементу result, у которого есть вложенный тег page с атрибутом is-default равным 1.
- Условие для системных страниц match="result[not(page)]" говорит о том, что этот шаблон надо применять к элементу result, у которого нет вложенного тега page — это правило верно для системных страниц.
- Условие match="result[//property[@name = ‘my_property’]]" говорит о том, что этот шаблон надо применять к элементу result, у которого есть поле с названием my_property.

В случае, если открытая страница не подходит ни под одно из условий, перечисленных в квадратных скобках, будет выбран первый шаблон match="result", так как это будет единственный подходящий шаблон. Таким образом, мы предусмотрели значение по умолчанию.

В случае, если несколько условий шаблонов оказываются с одинаковым приоритетом, будет выбран шаблон, который встретился в таблице стилей последним.

Образцы выбора

Во всех примерах, рассмотренных выше, мы выбрали элемент result исключительно для наглядности. В инструкции <xsl:apply-templates> в атрибуте select мы можем указать любые другие доступные элементы, при этом воспользовавшись все тем же способом записи: указав пути и условия.

Таким образом, мы будем оперировать уже 2 различными образцами: образец выбора и образец для применения шаблона.

Например, следующие три шаблона выведут список всех полей для открытой страницы, кроме поля типа «выпадающий список». Для случая с полем с HTML-кодом[1] будет задействован шаблон без экранирования тегов:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
         </head>
         <body>

            <h1><xsl:value-of select="result/@header"/></h1>

            <dl>
               <xsl:apply-templates select="//property[not(@type
               = 'relation')]"/>
            </dl>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="property">
      <dt><xsl:value-of select="title"/></dt>
      <dd><xsl:value-of select="value"/></dd>
   </xsl:template>


   <xsl:template match="property[@type = 'wysiwyg']">
      <dt><xsl:value-of select="title"/></dt>
      <dd><xsl:value-of select="value" disable-output-escaping="yes"/></dd>
   </xsl:template>

</xsl:stylesheet>
```

Пример 14. Использование образцов при выборе элементов

[1] Про типы полей UMI.CMS можно прочитать в пользовательской документации: http://help.umi-cms.ru/field_types.htm

Условие выбора select="//property[not(@type = ‘relation’)]" выбирает все свойства, кроме type = ‘relation’, а образец match="property[@type = ‘wysiwyg’]" будет обрабатывать поля с HTML-кодом.

Важно: выбор элементов и применение к ним шаблонов при помощи <xsl:apply-templates> будет происходить последовательно, в том порядке, в котором элементы находятся в исходном XML-документе.

Образцы и шаблоны в разработке сайта

Мы видим, что связка **<xsl:apply-templates> + <xsl:template>** и принцип связи образцов с шаблонами даже на этом этапе позволяют нам самим указывать шаблонизатору случаи, в которых надо применять тот или иной шаблон. Разрабатывая сайт таким образом, мы можем сконцентрироваться именно на наших задачах, и описывать их именно так, как нам нужно.

Естественно, вместо текста «это — такая-то страница» мы можем вывести разное содержимое для разных страниц, воспользовавшись, например, макросами и другими возможностями UMI.CMS. При этом мы будем использовать тот же самый принцип взаимодействия образцов и шаблонов. Как это сделать, мы поговорим далее.

Резюме:

- Мы убедились в том, что использование образцов и связанных с ними шаблонов в таблице стилей XSLT позволяет нам самим управлять условиями применения шаблонов.
- В качестве практического примера работы образцов и шаблонов, мы разобрали то, как можно при помощи одной инструкции <xsl:apply-templates> и набора шаблонов выводить разное содержимое на разных страницах сайта, ориентируясь на входящие XML-данные.
- Мы убедились в том, что образцы для шаблонов в XSLT задаются по тому же несложному принципу, что и адресация внутри XML-документа.
- Таким образом, мы видим, что используя XSLT в качестве шаблонизатора, мы сами управляем и шаблонами, и шаблонизатором, и решаем именно те и только те задачи, которые стоят перед нами при разработке сайта.

Использование протоколов в шаблонах

Еще раз вернемся к мысли, звучавшей в первой части: мы работаем с данными в формате XML. Однако в предыдущих главах мы рассматривали работу только с файлом в формате UMIData — в котором содержатся сведения о текущей запрошенной странице.

В ситуации, когда кроме содержимого этой страницы мы хотим вывести какие-либо другие данные, например меню, или список новостей — мы должны получить эти данные каким-то способом у системы и, обработав, включить в код, который будет выведен в браузер.

Для этих целей существует система протоколов UMI.CMS. Вызовы по протоколам совершаются из шаблонов и возвращают XML-данные, которые также обрабатываются шаблонизатором.

В таком случае формулу работы шаблонизатора можно видоизменить следующим образом:

Запрос из браузера ⟶ UMIData + XML-данные по протоколам + XSLT-шаблоны ⟶ HTML код в браузере

Или, еще проще, следующей диаграммой:

Рисунок 8. Формирование HTML-кода XSLT-шаблонизатором из нескольких источников

Протокол UData: результаты работы макросов

Откроем шаблон, который мы использовали в предыдущей главе, и допишем туда уже знакомую нам инструкцию <xsl:apply-templates>, однако в атрибуте select укажем document('udata://content/menu')/udata:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates
               select="document('udata://content/menu')/udata"/>
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>

            <xsl:apply-templates select="result"/>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

</xsl:stylesheet>
```

Пример 15. Подключение результатов работы макроса

Давайте теперь разберемся, что именно это означает.

udata://content/menu — это запрос по протоколу UData, который вернет результаты выполнения макроса content menu(). Если необходимо передать параметры, их следует дописать к запросу через "/".

Стоит запомнить: структуру результатов ответа любого макроса вы можете посмотреть в браузере, написав запрос вида: http://ваш_сайт/udata/имя_модуля/имя_метода/параметр1/параметр2...

Этот вызов может быть запрещен настройками системы (см. главу «Общие замечания по протоколам UMI.CMS»).

Функция **document()** подключает полученные данные в виде XML-документа. В этом XML-документе мы выбрали элемент udata и указали, что надо применить к этим данным подходящие шаблоны, при помощи инструкции <xsl:apply-templates>.

Обработка результатов макросов

Помните, мы говорили о том, что после того как выбран элемент в <xsl:apply-templates>, XSLT-процессор должен будет искать подходящие шаблоны для обработки выбранного элемента? Наша задача – предоставить такой шаблон.

Добавим следующий код: <xsl:template match="udata[@module = 'content'][@method = 'menu']">

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates
               select="document('udata://content/menu')/udata"/>
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="result">
```

```
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'content'][@method = 'menu']">
      … здесь мы выведем меню …
   </xsl:template>

</xsl:stylesheet>
```

Пример 16. Обработка результатов работы макроса при помощи <xsl:template>

Теперь на страницах сайта вместо <xsl:apply-templates> с вызовом макроса будет надпись «… здесь мы выведем меню …».

Почему именно этот шаблон был выбран для обработки результатов макроса?

Все по тем же причинам: его условие соответствия говорит о том, что его надо применять ко всем элементам udata, у которых атрибут module равен content и атрибут method равен menu. Именно такой элемент udata и возвращает этот макрос.

Теперь наша задача — вывести значения, доступные внутри выбранного элемента udata. Если опять обратиться к сравнению с файловой системой, — в этом шаблоне мы зашли в папку "udata" и теперь нас интересуют элементы item.

Давайте воспользуемся инструкцией <xsl:apply-templates> еще раз и оформим меню в виде списка:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates
               select="document('udata://content/menu')/udata"/>
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
```

```
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'content'][@method = 'menu']">
      <ul>
         <xsl:apply-templates select="items/item"/>
      </ul>
   </xsl:template>

</xsl:stylesheet>
```

Пример 17. Выбор пунктов меню в результатах макроса при помощи <xsl:apply-templates>

На этот раз в атрибуте select мы не конкретизировали, какой именно элемент item выбрать. Как мы уже говорили, в этом случае будут выбраны **все** элементы, удовлетворяющие образцу "items/item".

> **Важно:** напомним, что выбор элементов и применение к ним шаблонов при помощи <xsl:apply-templates> будет происходить последовательно, в том порядке, в котором элементы находятся в исходном документе — в данном случае XML-документе ответа макроса.

С момента вызова <xsl:apply-templates select="items/item"> шаблонизатор будет искать шаблоны для каждого элемента item. Добавим их, дописав еще две инструкции <xsl:template>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
form">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates
               select="document('udata://content/menu')/udata"/>
```

```
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'content'][@method = 'menu']">
      <ul>
         <xsl:apply-templates select="items/item"/>
      </ul>
   </xsl:template>

   <xsl:template match="item">
      <li>
         <xsl:value-of select="."/>
      </li>
   </xsl:template>

   <xsl:template match="item[@status = 'active']">
      <li class="active">
         <xsl:value-of select="."/>
      </li>
   </xsl:template>

</xsl:stylesheet>
```

Пример 18. Обработка отдельных пунктов меню при помощи <xsl:template>

Условие match="item[@status = 'active']" будет применено для активного пункта меню (в ответе макроса он будет помечен атрибутом status со значением active). Для всех остальных пунктов меню будет применен шаблон match="item". Обозначение " . " (точка), указывает на текущий элемент, то есть item.

Мы не расставили ссылки. Их значения можно получить из атрибута link для элемента item и, воспользовавшись записью с фигурными скобками, подставить в атрибут href тега <a>. Для активного пункта меню мы ссылку ставить не будем:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Trans-
form">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates select="document('udata://content/
               menu')/udata"/>
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'content'][@method = 'menu']">
      <ul>
         <xsl:apply-templates select="items/item"/>
      </ul>
   </xsl:template>

   <xsl:template match="item">
      <li>
         <a href="{@link}"><xsl:value-of select="."/></a>
      </li>
   </xsl:template>

   <xsl:template match="item[@status = 'active']">
      <li class="active">
         <xsl:value-of select="."/>
      </li>
   </xsl:template>

</xsl:stylesheet>
```

Пример 19. Ссылки с неактивных пунктов меню — вывод значения при помощи { }

Режимы шаблонов

Давайте выведем теперь ленту новостей с анонсами. Для этого воспользуемся макросом news lastlist(). Как вызвать макрос мы уже знаем. Как связать с ним шаблон для обработки результатов — тоже.

Поэтому допишем <xsl:apply-templates select="document('udata://news/lastlist/sitenews')/udata"/> и два шаблона <xsl:template>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates
               select="document('udata://content/menu')/udata"/>
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>

            <div>
               <xsl:apply-templates
               select="document('udata://news/lastlist/sitenews')/udata"/>
            </div>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'news'][@method = 'lastlist']">
      <h2>Новости:</h2>
      <ul>
         <xsl:apply-templates select="items/item" mode="news"/>
      </ul>
   </xsl:template>

   <xsl:template match="item" mode="news">
      <li>
         <a href="{@link}"><xsl:value-of select="."/></a>
```

```
        </li>
    </xsl:template>

    <xsl:template match="udata[@module = 'content'][@method = 'menu']">
        <!-- код шаблона -->
    </xsl:template>

    <xsl:template match="item">
        <!-- код шаблона -->
    </xsl:template>

    <xsl:template match="item[@status = 'active']">
        <!-- код шаблона -->
    </xsl:template>

</xsl:stylesheet>
```

Пример 20. Вызов макроса и шаблоны для списка в ленте новостей с режимом 'news'

Как видите, мы вызвали макрос с параметром (имя ленты новостей), выбрали элемент udata и указали, что надо применить шаблоны. Далее мы создали шаблон, который обрабатывает элемент udata и внутри него точно также выбрали все элементы item.

Однако мы знаем, что поиск шаблонов для элементов item будет производиться среди всех доступных шаблонов. То есть, возможна путаница с шаблонами, обрабатывающими элементы меню. Специально, чтобы этого не произошло, мы добавили **«режим»** в обработку — атрибут mode="news". Теперь для этих элементов item будут приняты в рассмотрение только шаблоны, у которых назначен такой же режим. Название режима мы задаем сами так, как нам удобно.

Если посмотреть на результаты работы макроса news lastlist(), то можно увидеть, что текста анонсов там нет. Это означает, что их надо получить еще одним вызовом. Для того, чтобы это сделать, познакомимся с еще одним протоколом UMI.CMS.

Протокол UPage: страницы сайта и их свойства

По этому протоколу мы можем получить в виде XML-документа любую страницу сайта.

Помните, мы уже проделывали это в первой части, когда рассматривали объекты и страницы сайта в виде XML?

Теперь необходимо включить эти данные в обработку в таблице стилей XSLT. И мы уже знаем, как это сделать — воспользоваться все той же функцией **document()**.

nb **Стоит запомнить:** поскольку нас интересует только анонс, мы можем воспользоваться сокращенной записью для протокола UPage. Для этого необходимо через " . " (точку) дописать имя свойства, которое мы хотим получить.

Например: upage://7.anons

Вспомним — мы хотели вывести анонсы для каждой новости из списка. И для этого необходимо передать id страницы новости в запрос по протоколу UPage в том же шаблоне, где мы выводим заголовок. Для формирования сложного запроса воспользуемся функцией **concat()**.

Добавим в шаблон следующий код:
<xsl:value-of select="document(**concat**('upage://', @id, '.anons'))//value"/>:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
            <meta name="description" content="{result/meta/description}"/>
            <meta name="keywords" content="{result/meta/keywords}"/>
         </head>
         <body>
            <div>
               <xsl:apply-templates select="document('udata://content/
               menu')/udata"/>
```

```
            </div>

            <h1><xsl:value-of select="result/@header"/></h1>
            <xsl:apply-templates select="result"/>

            <div>
               <xsl:apply-templates
               select="document('udata://news/lastlist/(news)')/udata"/>
            </div>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="result">
      <p>модуль: <xsl:value-of select="@module"/></p>
      <p>метод: <xsl:value-of select="@method"/></p>
   </xsl:template>

   <xsl:template match="udata[@module = 'news'][@method = 'lastlist']">
      <h2>Новости:</h2>
      <ul>
         <xsl:apply-templates select="items/item" mode="news"/>
      </ul>
   </xsl:template>

   <xsl:template match="item" mode="news">
      <li>
         <a href="{@link}"><xsl:value-of select="."/></a>
         <xsl:value-of select="document(concat('upage://',
         @id, '.anons'))/udata/property/value"
         disable-output-escaping="yes"/>
      </li>
   </xsl:template>

   <xsl:template match="udata[@module = 'content'][@method = 'menu']">
      <!-- код шаблона -->
   </xsl:template>

   <xsl:template match="item">
      <!-- код шаблона -->
   </xsl:template>

   <xsl:template match="item[@status = 'active']">
      <!-- код шаблона -->
   </xsl:template>

</xsl:stylesheet>
```

Пример 21. Вызов по протоколу UPage и построение сложных запросов при помощи concat()

Как можно видеть, для каждого элемента item в шаблоне производиться вызов по протоколу UPage, в который фукция **concat()** подставляет атри-

бут id и строку '.anons', и инструкция value-of выводит значение элемента value.

Для того чтобы отменить экранирование тегов можно добавить атрибут disable-output-escaping="yes".

Точно таким же образом мы можем получить любое другое свойство страниц сайта и использовать их в шаблонах.

Для работы с объектами существует аналогичный протокол.

Протокол UObject: объекты сайта и их свойства

По этому протоколу мы можем получить в виде XML-документа любой объект сайта. Мы также уже проделывали это в первой части: когда рассматривали объекты и страницы сайта в виде XML.

Так же как и в протоколе UPage мы можем использовать сокращенную запись, для получения отдельного свойства.

Например, если напишем в адресной строке следующее: uobject/14.login, мы получим логин пользователя с id = 14

Аналогично, при помощи функций document() и, если необходимо, при помощи concat() мы можем включать полученные XML-данные в обработку в шаблоне.

Другие протоколы UMI.CMS

Кроме UData, UPage и UObject в UMI.CMS можно воспользоваться еще несколькими протоколами, например:

UHttp: данные из внешнего XML-ресурса

Например, если у нас есть XML-ресурс, расположенный по адресу:
http://export.yandex.ru/weather/?city=26063 (погода в Петербурге)

И сами данные выглядят следующим образом:

```
<weather>
   <date>
      <day>10</day>
      <month>7</month>
      <year>2010</year>
   </date>
   <city>Санкт-Петербург</city>
   <country>Россия</country>
   <weather_type>Ясно</weather_type>
   <image>http://weather.yandex.ru/i/7.gif</image>
   <image2>http://weather.yandex.ru/i/7.png</image2>
   <temperature>+25</temperature>
   <pressure>763</pressure>
   <dampness>47</dampness>
</weather>
```

Пример 22. Содержимое внешнего XML-ресурса: погода за 10.07.2010

То мы можем включить эти данные в обработку в таблице стилей при помощи той же функции document() и применить к ним шаблоны, чтобы оформить вывод так, как нам это необходимо. Например, так:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <meta></meta>
         </head>
         <body>
            <h1>Погода в Петербурге сегодня:</h1>
            <xsl:apply-templates
            select="document('uhttp://export.yandex.ru/weather/?city=26063')/weather" />
         </body>
      </html>
   </xsl:template>

   <xsl:template match="weather">
      <p>Дата: <xsl:value-of select="date/day"/>.<xsl:value-of
      select="date/month"/>.<xsl:value-of select="date/year"/></p>

      <p><xsl:value-of select="weather_type"/>:<img src="{image}"/></p>

      <p>Температура: <xsl:value-of select="temperature"/></p>
```

```
    </xsl:template>

</xsl:stylesheet>
```

Пример 23. подключение и обработка внешних XML-данных при помощи document() и <xsl:apply-templates>

USel: выборки из базы данных UMI.CMS

Этот протокол позволяет делать выборки объектов и страниц по заданным критериям из базы данных UMI.CMS без написания SQL-запросов.

Запросы производятся по шаблонам, описанным в XML-формате. Например:

```
<?xml version="1.0" encoding="utf-8"?>
<selection>
    <target result="pages">
        <type module="catalog" method="object" />
    </target>
    <property name="best_offers" value="1" />
    <sort>rand()</sort>
    <limit page="0">{limit}</limit>
</selection>
```

Пример 24. Вариант шаблона USel с параметром limit

Дословно этот запрос означает следующее: выбрать страницы, возвращаемые модулем «Каталог» и методом object (это страницы объектов каталога), у которых свойство с name = ‘best_offers’ равно 1, отсортировать их в случайном порядке и вывести количество, определяемое переданным параметром limit.

Ответ системы может выглядеть вот так:

```
<udata generation-time="0.034025">

    <page id="6" parentId="4"
    link="/shop/dvd_tehnika/dvd_pleery/sony_ps6750/" is-active="1"
    object-id="27597" type-id="828" update-time="1277161583"
    alt-name="sony_ps6750" xlink:href="upage://6">
        <basetype id="6" module="catalog"
        method="object">Объекты каталога</basetype>
        <name>Sony PS-6750</name>
    </page>

    <page id="52" parentId="49"
    link="/shop/televizory/cherno-belye_televizory/soviet_ba/" is-active="1"
    object-id="27644" type-id="829" update-time="1277161609"
    alt-name="soviet_ba" xlink:href="upage://52">
        <basetype id="6" module="catalog"
        method="object">Объекты каталога</basetype>
```

```
      <name>Soviet BA</name>
   </page>

   <page id="42" parentId="41"
   link="/shop/televizory/cvetnye_televizory/sovet_tc1/" is-active="1"
   object-id="27633" type-id="829" update-time="1277161602"
   alt-name="sovet_tc1" xlink:href="upage://42">
      <basetype id="6" module="catalog"
      method="object">Объекты каталога</basetype>
      <name>Sovet TC1</name>
   </page>

   <total>8</total>
</udata>
```

Пример 25. XML-данные, полученные по протколу USel — выборка по критерию

Как видите, можно создать при помощи модуля «Шаблоны данных» какое-нибудь свойство у объектов и страниц, вписать туда значение, и выбирать из всего множества объектов системы только те, у которых это свойство равно нужному нам значению. При этом в шаблон запроса можно передавать параметры, что позволяет использовать один шаблон запроса более гибко.

nb **Стоит запомнить:** чтобы посмотреть, что именно вернет запрос по шаблону USel, можно набрать в окне браузера:

http://ваш_сайт/usel/имя_шаблона_запроса

Этот вызов может быть запрещен настройками системы (см. главу «Общие замечания по протоколам UMI.CMS»).

Все шаблоны запросов должны лежать в папке ~/usels/, быть в кодировке UTF-8 и иметь расширение ".xml".

Аналогично, мы можем включить эти данные в шаблонах при помощи той же функции document() и применить к ним шаблоны. Например, для приводившегося выше шаблона запроса так:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <meta></meta>
         </head>
         <body>
```

```
            <h1>Специальные предложения:</h1>

            <xsl:apply-templates select="document('usel://special-
            offers?limit=5')/udata"/>

         </body>
      </html>
   </xsl:template>

   <xsl:template match="udata">
      <p>Всего:
         <xsl:value-of select="total"/>
      </p>
      <ul class="special-offers">
         <xsl:apply-templates select="page"/>
      </ul>
   </xsl:template>

   <xsl:template match="page">
      <li>
         <a href="{@link}">
            <xsl:value-of select="name"/>
         </a>
         <a href="{@link}">
            <img src="{document(concat('upage://', @id, '.photo'))//val-
            ue}"/>
         </a>
      </li>
   </xsl:template>

</xsl:stylesheet>
```

Пример 26. Обработка данных, полученных по протоколу USel (см. Пример 24).

Мы передали в шаблоны выборки параметр limit=5, что даст нам 5 случайных объектов каталога, с отмеченной опцией 'best_offers'. В этом коде задействован также протокол UPage, который понадобился, чтобы вывести изображение для объекта каталога. Ранее мы обсуждали, что аналогичным способом можно получить любое другое свойство.

Подробно о структуре построения запросов можно посмотреть в документации разработчика сайта[1].

[1] http://help-dev.umi-cms.ru/chapter.XSLTTemplates.usel.html

UFs: данные о файлах и папках

Этот протокол позволяет получить данные о файлах или папках в виде XML.

Например, если ввести запрос в адресной строке: http://ваш_сайт/ufs/images/design, то будет показано содержимое папки design из папки images:

```
<udata path="_путь_\images\design" generation-time="0.001187">
   <directory name="bg"/>
   <directory name="item"/>
   <directory name="payment"/>
   <file name="icons.gif" ext="gif" size="2951"
   create-time="1277161530" modify-time="1277161530"/>
   <file name="logo.gif" ext="gif" size="1878"
   create-time="1277161530" modify-time="1277161530"/>
   <file name="nofoto.jpg" ext="jpg" size="23871"
   create-time="1277161530" modify-time="1277161530"/>
</udata>
```

Пример 27. Данные о файлах и вложенных папках в виде XML для папки ~/images/design

Аналогично, мы можем включить эти данные в шаблонах при помощи той же функции document() и применить к ним шаблоны. Например так:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8"/>

   <xsl:template match="/">
      <html>
         <head>
            <meta></meta>
         </head>
         <body>
            <h1>Содержимое папки design:</h1>
            <xsl:apply-templates select="document('ufs://images/design')/udata" />
         </body>
      </html>
   </xsl:template>

   <xsl:template match="udata">
      <h2><xsl:value-of select="@path"/></h2>
      <ul>
         <xsl:apply-templates/>
      </ul>
   </xsl:template>

   <xsl:template match="directory">
      <li class="directory">
```

```
            <xsl:value-of select="@name"/>
        </li>
    </xsl:template>

    <xsl:template match="file">
        <li class="file">
            <xsl:value-of select="@name"/>
            (<xsl:value-of select="@size"/> bytes)
        </li>
    </xsl:template>

</xsl:stylesheet>
```

Пример 28. Шаблоны для обработки списка папок и файлов

Этот код выведет список папок и файлов, причем папки будут оформлены по своему шаблону, файлы — по своему. <xsl:apply-templates/> без атрибутов означает: **выбрать все дочерние элементы от текущего** (в данном случае — дочерние элементы udata).

У этого протокола есть несколько управляющих параметров. Кроме того, он позволяет выводить информацию об отдельном файле. Подробное описание протокола можно посмотреть в документации разработчика сайта[1].

Общие замечания по протоколам UMI.CMS

Мы видели, что структуру данных, которые возвращают протоколы UMI.CMS мы можем в любой момент просмотреть в окне браузера, если это разрешено в config.ini.

Разрешение доступа по протоколам через HTTP

Для того, чтобы открыть доступ к какому-либо протоколу по HTTP, необходимо это явно указать в файле config.ini, добавив следующие опции для каждого протокола в секцию **[streams]**:

udata.http.allow = "1"
upage.http.allow = "1"
uobject.http.allow = "1"
uhttp.http.allow = "1"

usel.http.allow = "1"
ufs.http.allow = "1"

Когда разработка сайта закончена и следует закрыть доступ к некоторым данным, поставьте значение 0, для протоколов, доступ к которым надо закрыть, либо удалите соответствующие строки.

Примеры вызовов

Приведем примеры вызова из адресной строки для всех обсуждавшихся протоколов:

- http://адрес_сайта/udata/catalog/getObjectsList/notemplate/(/market/hamsters/)/10/0 — отобразит в окне браузера результаты макроса catalog getObjectsList(), с переданными четырьмя параметрами.
- http://адрес_сайта/upage/7 — отобразит в окне браузера данные о странице с id = 7.
- http://адрес_сайта/uobject/27529.image — отобразит в окне браузера данные в поле «изображение баннера» для баннера с id 27529.
- http://адрес_сайта/uhttp/blogs.yandex.ru/search.rss?text=umi.cms — отобразит в окне браузера ленту rss с результатами поиска "umi.cms" в блогах Yandex.
- http://адрес_сайта/usel/special-offers/10/5?limit=3 — отобразит в окне браузера результаты запроса по шаблону ~/usels/special-offers.xml с переданными параметрами 10, 5 и limit=3.
- http://адрес_сайта/ufs/images/design/bg?depth=1 — отобразит в окне браузера содержимое папки images/design/bg с глубиной вложенности 1.

Как мы неоднократно говорили ранее, все эти XML-данные могут быть включены в обработку в шаблонах, тогда как в браузере мы можем изучить их структуру, чтобы грамотно написать шаблоны.

Включение этих вызовов в шаблоны будет выглядеть так:

- document('udata://catalog/getObjectsList/notemplate/(/market/hamsters/)/10/0')
- document('upage://7')
- document('uobject://27529.image')
- document('uhttp://blogs.yandex.ru/search.rss?text=umi.cms')
- document('usel://special-offers/10/5?limit=3')
- document('ufs://images/design/bg?depth=1')

Рекомендуемая форма запроса

В случае если вы подключаете и применяете шаблоны при помощи <xsl:apply-templates>, рекомендуется выбирать корневой элемент (во всех случаях, кроме UHttp — это элемент udata):

- <xsl:apply-templates select="document('udata://content/menu')**/udata**" />
- <xsl:apply-templates select="document('uhttp://blogs.yandex.ru/search.rss?text=umi.cms')**/rss**" />

Так как иначе к результатам запроса (это тоже XML-документы и у них есть «корень») будет применен шаблон с **match="/"**, который мы используем в самом начале. Это, скорее всего, нежелательно[1].

Кеширование протоколов

Мы можем использовать кеширование для вызовов по этим протоколам. Для этого достаточно дописать "*?expire=время_в_секундах*" после вызова по протоколу:

xsl:apply-templates select="document('udata://content/menu?expire=600')/udata" />

[1] См. также главу «Итерации и рекурсии»

В этом случае, в течении заданного количества секунд, система не будет совершать реальных вызовов по протоколам, а будет брать данные из кеша.

Дополнительно: получение данных в формате JSON

Для получения всех этих данных в формате JSON (кроме протокола UHttp) следует к вызовам по протоколу дописать ".json"

Резюме:

- Мы видим, что система протоколов позволяет включать любые XML-данные в обработку при выводе страниц сайта — такие как результаты работы макросов, свойства страниц, объектов и другие, вплоть до внешних XML-источников.
- Подключение документов по протоколам производится всего одной функцией document(), а при помощи функции concat() можно динамически строить запросы по протоколам.
- Для написания шаблонов обработки результатов удобно использовать те же принципы сопоставления по образцу.
- Мы узнали, что можно использовать режимы в шаблонах, чтобы определить группу шаблонов, обрабатывающих тот или иной образец.

Дополнительные сведения об XSLT

В главе «Основные сведения об XSLT», мы рассмотрели главные принципы языка XSLT, понимание которых гарантирует, что знакомство с этой технологией состоялось. Мы также рассмотрели эти принципы в действии, на примере использования XSLT в качестве языка шаблонизатора UMI.CMS, и обсудили, как эти принципы можно использовать в разработке сайта.

Таким образом, сложный проект на UMI.CMS будет отличаться от простого — лишь большим количеством шаблонов и большим количеством запросов по протоколам, тогда как принципы остаются прежними.

Сейчас мы обсудим то, как можно оптимизировать некоторые моменты разработки и заодно познакомимся с оставшимися двумя принципами языка XSLT.

Таблицы стилей в нескольких файлах: <xsl:include>

Для того чтобы было легче ориентироваться в шаблонах большого проекта, рекомендуется разбить таблицу стилей по какому-либо принципу.

Для этого рекомендуется воспользоваться инструкцией <xsl:include>, указав в атрибуте href путь до файла. Например, так:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="utf-8"/>

      <xsl:include href="modules/content.xsl" />
      <xsl:include href="modules/news.xsl" />
      <xsl:include href="modules/users.xsl" />
      <xsl:include href="modules/catalog.xsl" />
      <xsl:include href="modules/emarket.xsl" />
      <xsl:include href="modules/photoalbum.xsl" />

      <xsl:template match="/">
         <html>
            <head>
               <title><xsl:value-of select="result/@title"/></title>
            </head>
            <body>
```

```
                <h1><xsl:value-of select="result/@header"/></h1>
                <div class="content">
                   <xsl:apply-templates select="result"/>
                </div>
             </body>
          </html>
       </xsl:template>

</xsl:stylesheet>
```

Пример 29. Подключение дополнительных таблиц стилей при помощи <xsl:include>

Все перечисленные файлы в приведенном примере должны находиться в папке modules, на уровень ниже вызывающей таблицы стилей. Каждый подключаемый файл должен быть также таблицей стилей XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <xsl:template match="…">
      <!-- код шаблона -->
   </xsl:template>

   <!-- ... и так далее -->

</xsl:stylesheet>
```

Пример 30. Внешняя таблица стилей

Перед тем как приступить к обработке входящих XML-данных, процессор XSLT объединит все эти таблицы стилей и будет работать с ними как с единым набором шаблонов.

Итоговый порядок расположения шаблонов определяется порядком инструкций <xsl:include>: упрощенно можно считать, что набор подключаемых шаблонов будет «вставлен» вместо этой инструкции.

Разработчик может сам решать, какие шаблоны в какие файлы помещать — если придерживаться единого принципа, найти необходимый шаблон в дальнейшем не составит труда.

Например, при разработке сайта на UMI.CMS очень удобно сгруппировать шаблоны по отдельным модулям: так файл content.xsl будет содержать

всю обработку страниц контента, news.xsl — отвечает за новости, и так далее. Но можно воспользоваться любой другой системой, если она кажется привычнее или удобнее.

Параметры и переменные в XSLT

Параметры и переменные в языке XSLT позволяют упростить некоторые действия; например, при разработке сайта, можно таким образом снизить число обращений по протоколам для подключения XML-данных.

Однако в использовании параметров и переменных в XSLT есть своя специфика. Давайте познакомимся с еще одним положением языка XSLT.

Код XSLT свободен от побочных эффектов (сторонние эффекты, side-effects)

Что такое побочные эффекты можно почитать самостоятельно[1]. Нас сейчас интересует одно из следствий из этого положения, касающееся переменных и параметров в XSLT: **переменные и параметры в XSLT — неизменяемые**. То есть значение, которое было получено при создании, в дальнейшем не может быть изменено.

В чем разница между параметрами и переменными в XSLT?

- Переменная создается и используется в текущем контексте.
- Параметр принимает значение извне и используется в текущем контексте.

[1] http://ru.wikipedia.org/wiki/Побочный_эффект_(программирование)
http://en.wikipedia.org/wiki/Side_effect_(computer_science)

Переменные: <xsl:variable>

Переменную можно задать локально или глобально:

- Локальная переменная будет доступна только внутри элемента, в котором она создана — в подавляющем большинстве случаев — это элемент <xsl:template>[1]. Однако ее можно передать в качестве параметра при вызове <xsl:apply-templates>.
- Глобальные переменные будут доступны во всех шаблонах, однако создавать их надо внутри элемента <xsl:stylesheet> («на одном уровне» с <xsl:template>).

Давайте рассмотрим следующий пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" indent="yes"/>

   <!-- глобальная переменная -->
   <xsl:variable name="pid" select="/result/@pageId"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
         </head>
         <body>
            <h1><xsl:value-of select="result/@header"/></h1>
            <div class="content">
               <xsl:apply-templates select="result"/>
            </div>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="result[@module = 'news'][@method = 'rubric']">
      <xsl:value-of select="$pid"/>
      то же, что и <xsl:value-of select="@pageId"/>

      <!-- локальная переменная -->
      <xsl:variable name="newslist"
      select="document(concat('udata://news/lastlist/', @pageId))/udata"/>
```

[1] Другие варианты, скорее всего, при разработке сайтов не понадобятся, но можно посмотреть подробнее здесь: http://www.w3.org/TR/xslt#variables

```
      <xsl:apply-templates select="$newslist">

      <!-- постраничный вывод -->
      <xsl:apply-templates select="document(concat('udata://system/numpages/',
      $newslist/total, '/', $newslist/per_page))/udata"/>
   </xsl:template>

</xsl:stylesheet>
```

Пример 31. Глобальные и локальные переменные

Как можно видеть, мы объявили глобальную переменную $pid (идентификатор открытой страницы) и во втором шаблоне — локальную переменную $newslist.

Вместо того чтобы совершать несколько вызовов по протоколам, мы вызвали один раз макрос news lastlist() и поместили в локальную переменную $newslist элемент udata из XML-ответа макроса. Теперь в этой переменной содержится фрагмент XML-данных. Поэтому мы обратились к значению вложенных элементов переменной $newslist при помощи путей и передали их другому макросу — system numpages().

Глобальную переменную мы вывели, указав ее имя, и заодно убедились, что в шаблоне она по-прежнему возвращает атрибут pageId для тега <result>.

Параметры <xsl:param>

Аналогично, параметры можно задать локально или глобально.

- Локальный параметр принимается внутри шаблона (внутри элемента <xsl:template>) и доступен только там; для того, чтобы передать параметр в шаблон, необходимо воспользоваться инструкцией <xsl:with-param>.
- Глобальные параметры принимаются всей таблицей стилей, доступны во всех шаблонах, аналогично — создавать их надо внутри элемента <xsl:stylesheet> («на одном уровне» с <xsl:template>).

Рассмотрим следующий пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" indent="yes"/>

   <xsl:param name="param" select="0"/>

   <xsl:template match="/">
      <html>
         <head>
            <title><xsl:value-of select="result/@title"/></title>
         </head>
         <body>
            <h1><xsl:value-of select="result/@header"/></h1>
            <div class="content">
               <xsl:apply-templates select="result"/>
            </div>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="result[@module = 'news'][@method = 'rubric']">
      <xsl:apply-templates
      select="document(concat('udata://news/lastlist/', @pageId))/udata">
         <xsl:with-param name="pid" select="@pageId"/>
      </xsl:apply-templates>
   </xsl:template>

   <xsl:template match="udata[@module = 'news'][@method = 'lastlist']">
      <xsl:param name="pid"/>
      $pid: <xsl:value-of select="$pid"/>
      $param: <xsl:value-of select="$param"/>
   </xsl:template>

</xsl:stylesheet>
```

Пример 32. Глобальные и локальные параметры

Если просто запросить страницу с лентой новостей, $param получит значение по умолчанию — '0' (мы указали select="0") и будет доступен во всех шаблонах.

Если запрос будет выглядеть, например, так:
http://адрес_сайта/название_ленты/?param=100, то значение будет передано в таблицу стилей, и параметр $param будет равен '100'. То есть, таким

образом, мы можем передать значения извне (из URL) и использовать их в шаблонах.

Кроме того, в этом шаблоне выводится параметр $pid, переданный из предыдущего шаблона при помощи <xsl:apply-templates> и <xsl:with-param>.

Для того, чтобы параметр был доступен, его надо объявить в этом шаблоне при помощи <xsl:param>. Это необходимо сделать непосредственно после элемента <xsl:apply-templates>, указав в атрибуте name то же название, что и было передано при помощи <xsl:with-param>.

Итерации и рекурсии

Настало время познакомиться с последним положением языка XSLT.

Основой преобразования XSLT являются итерации и рекурсии

С итерациями мы уже встречались, когда выводили список полей у страницы, или обрабатывали список элементов из результатов макроса. При помощи <xsl:apply-templates> в атрибуте select был задан образец выбора, а потом элементы были обработаны последовательно в том порядке, в каком они встретились в исходном документе.

Помните, что поиск подходящих шаблонов совершается по всем доступным шаблонам? Как следствие, шаблон, внутри которого совершается вызов инструкции <xsl:apply-templates> также может попасть в список, и вызвать, таким образом, сам себя.

В тех случаях, когда это нежелательно, рекомендуется воспользоваться другим режимом для шаблонов.

Однако в некоторых ситуациях этот подход может заметно сэкономить количество кода. Рассмотрим следующий пример:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="UTF-8" indent="yes"/>

   <xsl:template match="/">
      <html>
      <head>
         <title><xsl:value-of select="result/@title"/></title>
      </head>
      <body>
         <h1><xsl:value-of select="result/@header"/></h1>
         <div class="content">
            <xsl:apply-templates select="result"/>
         </div>
      </body>
      </html>
   </xsl:template>

   <xsl:template match="result[@module = 'content'][@method
   = 'notfound' or @method = 'sitemap']" >
      <xsl:apply-templates
      select="document('udata://content/sitemap')/udata/items"/>
   </xsl:template>

   <xsl:template match="items">
      <ul>
         <xsl:apply-templates select="item"/>
      </ul>
   </xsl:template>

   <xsl:template match="item">
      <li>
         <a href="{@link}"><xsl:value-of select="@name" /></a>
         <xsl:apply-templates select="items"/>
      </li>
   </xsl:template>

</xsl:stylesheet>
```

Пример 33. Использование рекурсии для построения карты сайта

В этом примере для страницы с картой сайта и для страницы 404 (страница не найдена), указано, что надо отобразить результаты макроса content sitemap() (макрос, возвращающий список элементов для карты сайта).

Ответ макроса выглядит примерно следующим образом:

```
<udata module="content" method="sitemap" generation-time="0.018748">
   <items>
      <item id="1" link="/" name="Добро пожаловать" xlink:href="upage://1"/>
      <item id="2" link="/talks/" name="Форум" xlink:href="upage://2"/>
      <item id="12" link="/vse_novosti/" name="Все новости"
      xlink:href="upage://12">
         <items>
            <item id="13" link="/vse_novosti/politicheskie_novosti/"
            name="Политические новости" xlink:href="upage://13"/>
            <item id="19" link="/vse_novosti/novosti_ekonomiki/"
            name="Новости экономики" xlink:href="upage://19"/>
         </items>
      </item>
      <item id="26" link="/contacts/" name="Обратная связь"
      xlink:href="upage://26"/>
      <item id="27" link="/umicms/" name="FAQ" xlink:href="upage://27"/>
      <item id="40" link="/downloads/" name="Скачать"
      xlink:href="upage://40">
         <items>
            <item id="41" link="/downloads/php_manual/" name="PHP Manual"
            xlink:href="upage://41"/>
         </items>
      </item>
      <item id="42" link="/obychnaya_stranica/" name="Обычная страница"
      xlink:href="upage://42"/>
      <item id="43" link="/market/" name="Каталог товаров"
      xlink:href="upage://43"/>
   </items>
</udata>
```

Пример 34. Карта сайт в виде XML — ответ макроса content sitemap()

То есть в нем уже содержатся все данные о страницах и иерархии, надо лишь преобразовать эти данные и вывести результаты в виде списка ссылок.

Мы выбрали первый элемент items, находящийся непосредственно в элементе udata и указали, что надо к нему применить шаблоны (select="document('udata://content/sitemap')/udata/items").

Будет выбран шаблон с match="items", внутри которого будет последовательно обработан каждый элемент item (итерация) по шаблону с match="item".

Однако в последнем шаблоне стоит еще одна инструкция <xsl:apply-templates>, которая указывает, что необходимо применить шаблоны к дочернему элементу items, для текущего item. Если он существует (в том случае, когда есть вложенные страницы), то будет совершен поиск шаблона — и снова найден шаблон с match="items", внутри которого будет произведена обработка вложенного элемента item.

Таким образом, при помощи двух последних шаблонов — с match="items" и match="item" будет рекурсивно обработано все XML-дерево ответа макроса, вне зависимости, сколько уровней вложенности в нем присутствует.

Аналогичным способом мы можем создать универсальные шаблоны для многоуровневого меню, положившись на XSLT и CSS.

Общие рекомендации по созданию шаблонов

Подведем некоторые итоги всего того, что мы обсуждали в качестве советов и предложений выше, а также постараемся предупредить некоторые ошибки, неизбежно возникающие у начинающих.

Много небольших шаблонов vs. один большой шаблон

Помните, что при выводе данных мы используем принцип разделения логики и представления?

Мы видели, что при правильном подходе, в результате должно было получиться много небольших шаблонов, каждый из которых будет описывать некую «сущность», например:

- Шаблон для страницы контента — страницу контента.
- Шаблон для всего меню (для ответа макроса) – блок меню.
- Шаблон для пункта меню и шаблон для активного пункта меню — элементы меню.
- Шаблон для страницы ленты новостей — страницу ленты новостей.
- Шаблон для списка новостей (для ответа макроса) — блок с лентой.
- Шаблон для отдельного элемента из списка новостей — новость в списке.
- И так далее.

Этот подход гарантирует, что мы легко можем ориентироваться в списке шаблонов, легко можем понять, что именно они делают, и, в случае необходимости, дописать несколько дополнительных шаблонов, не нарушая работу остальных. Проекты сами по себе получаются масштабируемыми.

Если кроме этого, мы сгруппируем шаблоны в отдельных файлах, напри-

мер, относящихся к отдельным модулям или даже действиям, ориентироваться в них будет еще проще.

Этот подход также сильно упрощает задачу связки шаблонов и CSS при верстке сайта: скорее всего, наблюдательные читатели уже заметили некоторое сходство в подходе у таблиц стилей XSLT и у таблиц стилей CSS. Именно поэтому верстальщики, как правило, с легкостью осваивают XSLT, и не испытывают с ним в дальнейшем никаких трудностей.

Соблюдение этих несложных правил, а также советов приведенных ниже (не поддаваться «соблазнам») позволяет очень быстро накопить набор готовых решений, которые с легкостью можно использовать из проекта в проект, лишь меняя стили CSS и основные шаблоны внешнего вида страниц.

Соблазны для начинающих: «вредные» инструкции

Мы специально не рассматривали пока некоторые инструкции, которые при самостоятельном изучении XSLT сразу бросаются в глаза как «знакомые». К сожалению, многие разработчики, «признав» их, оставляют попытки разобраться в ключевых принципах (таких как сопоставление по образцу, итерации и рекурсии), и сталкиваются с проблемами «медленности» и «неудобства». Об этом мы упоминали в самом начале разговора об XSLT.

Подробно эти инструкции мы разбирать не будем, лишь кратко прокомментируем.

Инструкция <xsl:if>
Задает условие. Соблазн — смешать обратно логику и представление.

Помните, каким образом мы применяли шаблоны для разных страниц сайта? Как вы видели, сам принцип использования образцов и шаблонов с выражениями XPath в большинстве случаев лишает необходимости задавать условия где-то еще.

Кроме того, посмотрите пожалуйста на предыдущую подглаву. Если мы оперируем «сущностями», каждую из которых мы описываем в отдельном шаблоне — то использование логических блоков, скорее всего, внесет путаницу в эту картину: мы не сможем связать шаблоны с реальными элементами сайта, снизится читаемость и понятность кода.

Вывод: используйте, только, если решение на <xsl:apply-templates> не подходит, не существует, или мы работаем не с элементом XML-дерева.

Инструкция <xsl:choose>
Выбирает из условий. Соблазн — смешать обратно логику и представление.

Все сказанное про <xsl:if> выше справедливо и для <xsl:choose>. Реально эта инструкция понадобится еще реже, чем <xsl:if>.

Инструкция <xsl:for-each>
Осуществляет итерацию по элементам по заданному образцу. Соблазн — использовать для «изобретения» циклов, вместо которых уже есть итерации и рекурсии при помощи <xsl:apply-templates>.

В разработке сайтов на UMI.CMS не понадобится. Все необходимое и так выполняет <xsl:apply-templates>.

Инструкция <xsl:for-each> известна также тем, что замедляет работу шаблонов XSLT.

Инструкция <xsl:call-template>
Вызывает шаблон с указанным именем. Соблазн — в связке с <xsl:with-param> использовать для «изобретения» «процедур» или «функций» на XSLT. Обычно активно используется с <xsl:if> и <xsl:choose>, что позволяет окончательно похоронить принцип разделения логики и представления, делает код шаблонов запутанным, медленным и нечитаемым.

Вы помните, что шаблоны и так вызываются при помощи сопоставления с образцами при помощи <xsl:apply-templates>? Причем вместо прямого

указания имени вызываемого шаблона, мы можем подстраивать ситуации в зависимости от обрабатываемых данных, добавить шаблон, который «дополнит» картину, обработает случай, ранее нам не попадавшийся — и так далее.

Вывод: используйте, если необходимо вывести какие-либо данные, не относящиеся к входящему XML-документу, либо не имеющие привязки к элементам дерева этого документа. Например, можно использовать, для вставки HTML-кода шапки сайта, или футера — если на всех страницах сайта они одинаковые.

XSLT и проблема входящих данных

Как правило, необходимость использования перечисленных выше инструкций продиктована плохой структурированностью входящих XML-данных. Получается, что в этих случаях, мы перекладываем на XSLT часть операций по работе с данными, которые, скорее всего, должны были быть выполнены заранее.

В таких ситуациях, как правило, и возникает иллюзия «сложности» и «медленности» технологии, о чем мы говорили в начале рассказа об XSLT.

Напомним, что в UMI.CMS используется принцип разделения логики и представления: поэтому XML-данные уже на входе структурированы максимально удобным образом, и не предполагают сложных и дорогостоящих операций на языке XSLT. Это позволяет не изучать все тонкости языка, а решать большинство задач, используя несколько базовых принципов и ограниченный набор инструкций и функций.

Отладка и тестирование шаблонов

Режим отладки

Для того чтобы можно было отследить ошибки незакрытых тегов, или ошибки иного рода, во время разработки сайта на UMI.CMS рекомендуется включить режим отладки:

Для этого необходимо установить параметр enabled="1" в секции [debug] в файле config.ini

Просмотр вызовов по протоколам с текущей страницы

Для того, чтобы увидеть список вызовов, которые были совершены при выводе этой страницы, можно добавить в адресной строке параметр **?showStreamsCalls**. В списке будут присутствовать только реальные вызовы, со временем, которое занял каждый вызов (закешированные вызовы в этот список не попадут):

```
<streams-call>
    <call generation-time="0.010091">udata://content/menu/0/2</call>
    <call generation-time="0.044340">udata://news/lastlist/(services)/notemplate/2</call>
    <call generation-time="0.000794">upage://6.anons</call>
    <call generation-time="0.000744">upage://7.anons</call>
    <call generation-time="0.047197">udata://news/lastlist/(news)/notemplate/2</call>
    <call generation-time="0.000242">udata://system/convertDate/1274885820/(d.m.Y)</call>
    <call generation-time="0.000806">upage://26.anons</call>
    <call generation-time="0.000231">udata://system/convertDate/1274885760/(d.m.Y)</call>
    <call generation-time="0.000746">upage://25.anons</call>
</streams-call>
```

Пример 35. Вызовы по протоколам совершенные с текущей страницы

Как видите, в приведенном списке вызываются макросы (протокол UData) и запрашиваются анонсы новостей (протокол UPage), мы также можем видеть с какими именно параметрами они вызваны.

Просмотр результатов работы макроса в контексте определенной страницы

Макросы UMI.CMS при запуске их из адресной строки при помощи вызова http://ваш_сайт/udata/имя_модуль/имя_макроса, возвращают XML вне контекста какой-либо страницы. Однако иногда возникает необходимость посмотреть ответ макроса в том виде, в каком он будет получен именно с этой страницы.

В таком случае можно воспользоваться вспомогательным шаблоном. Создайте в папке ~/xsltTpls/ файл под названием debug.xsl и скопируйте в него этот код:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" encoding="UTF-8" indent="yes"/>

   <xsl:param name="macro"/>

   <xsl:template match="/">

      <form method="post">
         <span style="font-weight:bold; margin-right:10px">udata://</span>
         <input type="text" name="macro" value="{$macro}"
         style="width:400px; margin-right:10px"/>
         <input value="Показать" class="submit" type="submit"/>
      </form>

      <textarea style="height:500px; width:900px;">
         <xsl:copy-of select="document(concat('udata://', $macro))"/>
      </textarea>

   </xsl:template>

</xsl:stylesheet>
```

Пример 36. Шаблон для просмотра результатов работы макроса в контексте страницы

Далее подключите этот шаблон к системе в настройках модуля "Структура" (см. «Создание и подключение XSLT-шаблона»). Теперь следует для интересующей нас страницы выбрать этот шаблон и открыть страницу на просмотр.

Вместо содержимого страницы вы увидите поле ввода, где можно ввести макрос с параметрами в виде имя_модуля/имя_макроса/параметр1/параметр2 и нажать “Показать". В области ниже будет показан результат работы этого макроса, запущенного именно с этой страницы.

Заключение

Подведем итоги:

- Мы начали с модели данных UMI.CMS, чтобы дать основные понятия, которыми мы оперируем в дальнейшем, и показали, как эта модель позволяет использовать формат XML для представления сущностей системы. На примере сущностей системы мы рассмотрели удобства формата XML, такие как наглядность, структурированность и естественность.
- Мы упомянули о том, что сайт на UMI.CMS можно рассматривать как набор XML-сервисов, поэтому роль шаблонизатора сводится к агрегации различных XML-данных и выводе необходимой информации на основании этих данных. При таком подходе язык XSLT оказывается идеальным языком шаблонизатора, работающего с XML.
- Мы обсудили первое положение языка XSLT (таблица стилей — это также документ XML), следствия из этого положения, представление XML-документов и адресацию внутри них при помощи путей XPath. Мы также показали, что это нам дает при разработке сайта на примере вывода любых значений из исходного документа UMIData.
- Мы обсудили второе положение языка XSLT (преобразование путем связывания образцов с шаблонами) и использование инструкций <xsl:apply-templates> и <xsl:template> — основного инструмента преобразования. Мы также показали, что это нам дает при разработке сайта на примере шаблонов для разных страниц сайта.
- Мы обсудили протоколы UMI.CMS и их роль в агрегации шаблонизатором XML-данных, показав также, что уже использование двух первых принципов XSLT позволяет обрабатывать эти данные и выводить их в том виде, в котором это необходимо.
- Мы затронули вопросы оптимизации разработки, такие как использование отдельных файлов для отдельных задач разработки сайта на UMI.CMS, использование переменных, параметров и рекурсий в таблицах стилей. В связи с этим мы рассмотрели оставшиеся два положения языка

XSLT (неизменяемые переменные и роль итераций и рекурсий).

- Мы кратко познакомились с общими рекомендациями и советами по созданию таблиц стилей, а также с некоторые «соблазнами», неизбежно возникающими перед разработчиками в начале знакомства с XSLT.
- И в завершение упомянули некоторые инструменты для отладки, которые можно задействовать при разработке сайтов на UMI.CMS.

Теперь, когда у вас есть эти знания, вы можете попробовать сделать сайт на UMI.CMS или изучить шаблоны тестовых демосайтов на примере локальной или триальной версии UMI.CMS[1]. Вы можете также ознакомиться с примером создания несложного сайта в 20 шагов с использованием принципов, изложенных в этой книге[2].

Удачи и успехов в разработке!

[1] http://www.umi-cms.ru/downloads/test_umicms/

[2] http://www.umi-cms.ru/support/docs/xslt-site-in-20-steps/

Для заметок