МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ОДЕССКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
имени И.И. МЕЧНИКОВА

Институт математики, экономики и механики
Кафедра математического обеспечения компьютерных систем

О.А. ГЕРЕНКО, О.И. РОЗНОВЕЦ, Е.А. ПЕНКО

Методическое пособие по курсу «Язык разметки XML. Часть 1»

Методическое пособие для студентов специальности «Компьютерные системы и сети» 4 курса дневного/заочного отделений

ОДЕССКИЙ НАЦИОНАЛЬНЫЙ
УНИВЕРСИТЕТ
имени И.И. МЕЧНИКОВА
2010

Методическое пособие по курсу «Язык разметки XML. Часть 1» для студентов специальности «Компьютерные системы и сети» 4 курса дневного/заочного отделений.

***Авторы**:*

О.А. ГЕРЕНКО,
старший преподаватель

О.И. РОЗНОВЕЦ,
ассистент

Е.А. ПЕНКО, ассистент

***Рецензенты**:*

А.В. КАМЕНЕВА,
кандидат технических наук, доцент

Л.А. ВОЛОЩУК,
кандидат технических наук, доцент

Рекомендовано к изданию Учёным советом
Института математики, экономики и механики
ОНУ имени И.И. Мечникова.
Протокол № 1 от 9 октября 2009 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Создание лабораторного практикума по курсу «Язык разметки XML» ставит своей целью:

- создание у студентов интереса к изучению курса «Язык разметки XML»;
- предоставление студентам необходимой информации по выполнению лабораторных работ;
- подготовка источников информации с программным обеспечением для установки и настройки всего необходимого для выполнения лабораторных работ;
- предоставление студентам примерных алгоритмов выполнения лабораторных работ.

В лабораторном практикуме продемонстрированы основные моменты выполнения следующих лабораторных и практических работ:

- статическая и динамическая информационная модель;
- основные конструкции языка XML;
- создание динамических web - страниц на основе языков xml, xsl;
- стилевые таблицы XSL, JavaScript и XML;
- пример построения гистограммы;
- DTD – определения (Document Type Definition);
- программная обработка XML документов с помощью XML- DOM;
- использование хранилища данных в виде XML-файла;
- Ajax, библиотека jQuery JavaScript.

Лабораторно-практические занятия предназначены для приобретения каждым студентом индивидуальных практических навыков по разработке web – технологий с использованием языка XML.

Лабораторная работа № 1. Статическая и динамическая информационная модель. Основные конструкции языка XML.

Цель работы:

Изучение принципов построения статической и динамической информационной модели. Изучение основных конструкций языка XML и правил оформления XML-документов.

Результат:

Синтаксически правильные XML-документы для выбранной предметной области.

Задание для самостоятельного выполнения

Представьте в виде древовидной структуры данные выбранной по желанию предметной области. Примерами предметных областей могут служить, например, «Издательство» (выпускающее газеты и журналы), «Научная конференция», «Кафедра вуза» или «Составление букетов».

Создайте XML-документ, использовав информационную модель, построенную в первой части лабораторной работы.

Методические указания

к первой части лабораторной работы

Моделирование данных и XML

Успех любого приложения XML зависит от того, насколько хорошо спроектированы фактически используемые документы XML: они должны быть способны не только нести информацию, которую люди передают друг другу сегодня, но и обладать достаточной гибкостью, чтобы учитывать будущие требования. Для этого необходимо рассмотреть следующие аспекты процесса проектирования:

Моделирование информации (понимание структуры и назначения информации, содержащейся в документах);

Проектирование документа (трансляция информационной модели в набор правил или схем для создания фактических документов);

Нотации схем (методы записи проекта документа, чтобы он был доступен как для обрабатывающего его программного обеспечения, так и для пользователя-человека).

Информационная модель — это описание используемой организацией информации, не зависящее от какой бы то ни было информационной технологии и определение таких моментов, как:

- каким образом она структурирована;
- что она означает;
- кому она принадлежит, и кто отвечает за ее своевременность и качество;
- откуда она берется и что происходит с ней в конце.

Моделирование информации имеет большое значение, потому что без модели нет информации, есть только данные. Информационная модель описывает назначение данных.

Любое информационное моделирование преследует две цели, которые не всегда бывает легко сочетать:

- *Получение абсолютно точных определений*
- *Эффективная коммуникация с пользователями*

Существуют два основных типа информационной модели: статическая и динамическая.

Статическая информационная модель

В *статической* модели описываются допустимые состояния системы, типы объектов в системе, их свойства и связи (вместе с этим определяются их имена). Достичь соглашения по именованию всех объектов очень важно, именно поэтому информационные модели XML иногда называют словарями.

При построении статической информационной модели необходимо пройти следующие этапы:

Этап 1. Идентификация понятий, присвоение им имен и их определение

Этап 2. Организация понятий в иерархию классов

Этап 3. Определение связей, множественности и ограничений

Этап 4. Добавление свойств для конкретизации деталей значений, связанных с объектами

Именование понятий

Для начала нужно составить список понятий, относящихся к системе. Иногда предлагают описать систему на бумаге и выделить все существительные. В любом случае этот этап, как правило, не представляет затруднений.

Далее, и это иногда требует больше времени, надо описать типы объектов. Описание термина должно быть точным, чтобы не возникало разногласий по существу определения. Ценность моделирования в том и заключается, что оно предотвращает появление потенциальных источников непонимания.

По завершении работы, вероятно, получится длинный список типов объектов, и у некоторых будут длинные имена. Следует выбирать такие имена, которые занятые в этом бизнесе люди смогут корректно понять и интерпретировать: дело в том, что они не всегда будут сверяться с написанными определениями.

Помимо именования типов объектов стоит определить также, каким образом можно идентифицировать индивидуальные экземпляры. Возможно, в этом виде бизнеса существует код, который следует знать, или написать его. Надо принимать во внимание, что в схемах кодирования в бизнесе часто обнаруживаются проблемы.

В конце этого этапа мы получим список типов тех объектов с именами и определениями, для которых достигнуто соглашение.

Таксономия

Таксономия — это термин, используемый в биологии для обозначения системы классификации; в информационном моделировании ее также называют иерархией типов (иногда ее называют еще онтологией). Перечислив и назвав типы объектов, их надо организовать в иерархическую схему классификации. Часто эти иерархические отношения возникают уже на этапе определения типов объектов.

Ключевой здесь является фраза, определяющая принадлежность (в английском языке — is или is kind of). Написав предложение вида "А есть

разновидность B" или "Каждое A есть B", вы определили отношения подтипов в вашей таксономии.

Иногда эти действия называются тестом "is a" ("есть", "представляет собой"). Однако будьте внимательны, поскольку эта конструкция используется также и для описания отношений между конкретным экземпляром и его типом, безопаснее писать этот тест в форме "is a kind of" ("представляет собой разновидность").

Идентификация подтипов бывает, полезна при проектировании документа, что более важно, она помогает лучше понять определения типов объектов.

Если вы занимались объектно-ориентированным программированием, то понимаете, как определять иерархии типов. Но программисты часто рассматривают классы объектов, прежде всего в терминах модулей функциональности внутри системы, а не понятий, которые они представляют в окружающем мире. Тогда для обозначения типов объектов используются глаголы, а не существительные - что неверно.

Итак, этап 2 сводится к организации типов объектов в иерархию типов.

Поиск связей

После того как объекты названы, в статическом информационном моделировании надо определить *связи*, существующие между ними. Связи (на языке UML они называются ассоциациями) можно показать просто сформулировав их в виде обычных предложений или их можно показать графически в виде диаграммы. Для диаграмм, описывающих связи между объектами, существует большое количество нотаций, каждый может выбрать предпочтительную для себя. Диаграммы следует делать предельно простыми и интуитивно понятными, сосредоточившись на ключевых сообщениях и оставив подробности текстовым документам, которые проще обслуживать.

Существует некая информация, которую надо знать о каждой связи:

Множественность связи показывает, сколько объектов каждого типа принимает в ней участие.

Связи типа "один-ко-многим": одна глава содержит много параграфов, один человек покупает много туристических поездок.

Связи типа "многие-ко-многим" также часто встречаются: один автор может написать несколько книг, но у книги также может быть несколько авторов.

Связи типа "один-к-одному".

При моделировании информации для окончательного представления XML особенно важным типом связей являются связи включения. Множественность этих связей всегда бывает "один-ко-многим" и "один-к-одному". Хотя четкого правила по поводу того, какие объекты образуют связи включения, не существует, можно иногда использовать правила обычного языка: глава содержит параграфы, курорт содержит отели, а отель содержит посетителей. В языке UML определено две формы связей включения. Первая - это *агрегации*, относительно свободное объединение объектов, позволяющее рассматривать их группу в течение некоторого времени как целое (например, туристическая группа, одни и те же люди могут в разное время входить в разные группы). Вторая форма - *композиция*. Это более строгая форма. Отдельные части целого не могут существовать независимо от него (например, комнаты в отеле не могут существовать независимо от отеля).

Найти подходящие имена для связей бывает нелегко. При записи связей полезно использовать полные фразы типа "отель расположен на курорте" или

"человек - это автор книги". Нам не надо использовать эти имена в тегах разметки XML, они присутствуют только в документации на систему.

Итак, мы определили связи, существующие между типами объектов в нашей модели.

Описание свойств

Типы объектов и связи формируют скелет статической информационной модели, свойства можно сравнить с плотью на костях. Свойства представляют собой простые значения, ассоциированные с объектами. У человека можно определить рост, вес, национальность и род занятий; отель имеет определенное количество комнат, этажность и ценовую категорию.

Не следует снова включать связи в список свойств объекта: расположение отеля не является его свойством, если мы уже промоделировали его как связь с курортом.

Главное, что нам надо знать о свойствах, - это тип их данных. Определен ли для них фиксированный диапазон значений, являются ли они числовыми, в каких единицах выражаются? Является ли свойство обязательным и есть ли у него значение по умолчанию?

В конце этапа 4 мы завершили формирование статической информационной модели: получили полное описание типов объектов в системе, их связей друг с другом и их свойств.

Динамическая информационная модель

Динамические модели описывают, что происходит с информацией: примерами таких моделей являются диаграммы рабочих процессов, потоков данных и жизненных циклов объектов. Динамические модели состоят примерно из таких утверждений: "Отделение патологии отправит результаты теста консультанту, отвечающему за пациента". Динамические модели описывают процесс обмена информацией: данные отправляются из одного места в другое с конкретной целью.

Для построения динамической модели можно воспользоваться различными программными системами (например, диаграммы рабочих процессов и диаграммы потоков данных можно построить, используя программу BPWin).

Модели рабочих процессов

Модели рабочих процессов заостряют внимание на роли людей и организаций в выполнении работы, хранение и обработка информации играют в них вторичную роль. Модель процесса описывает, например, что будет с путешественником, если на отдыхе с ним произойдет несчастный случай. Она определяет, за какие действия отвечает локальный представитель на курорте, агент в стране, где находи курорт и главный офис. В результате становится ясно, кто должен отвечать за организацию медицинской помощи, за перевозку туриста домой и за информирование родственников. Она может описывать различные формы, заполняемые и пересылаемые между участниками, и вообще не привлекать компьютерные системы. Модель процесса обычно фокусирует внимание на ролях, обязанностях и задачах каждого действующего лица системы (actor), а workflow-модель имеет дело с документами, передаваемыми между действующими лицами. Модели потоков данных.

Модели потоков данных

Модели потоков данных очень напоминают предыдущий тип, но здесь основное внимание уделяется информационным, а не бизнес-системам. Эта модель описывает хранилища данных (data stores), где информация находится постоянно (это может быть база данных в компьютере или просто кабинет с папками), процессоры, манипулирующие с этими данными, и потоки данных, передающие данные от одного процессора другому. Она активно использует статическую информационную модель: последняя описывает, что означают такие концепции, как путешественник или отель, но ничего не сообщает о том, где содержится информация. Напротив, из модели потоков данных становится ясно, что информация о туристической поездке находится в базе данных покупок до завершения поездки и оплаты всех счетов, после чего резюме этой информации передается в маркетинговую информационную систему, а все остальное - в архив.

Объектные модели

Объектные модели содержат как динамический, так и статический компоненты. Динамическая или поведенческая часть определения объекта сосредоточена на том, что может делать или делал каждый объект, представляя для этого набор операций или методов, описывающих его действия.

Жизненные циклы объекта

Жизненные циклы объекта (на языке UML это называется линиями жизни объекта) также заостряют внимание на индивидуальных объектах, но придерживаются более целостного подхода. Они описывают, что происходит с объектом на протяжении его жизни: как он создается, какие события с ним происходят, как он реагирует на эти события и какие условия приводят в конце к его разрушению.

Жизненные циклы объекта очень полезны для тестирования завершенности модели. Часто наблюдается тенденция к акцентированию внимания только на некоторых событиях за счет остальных. Пока мы не определим, каким образом каждый объект попадает в систему и как он удаляется из нее, полного понимания не будет.

Варианты использования

Варианты использования (use cases) анализируют выполнение специфических задач пользователя (например, человек, купивший туристическую путёвку, отменяет свой заказ). Вариант использования напоминает модель процесса, но в общем случае заостряет внимание на деятельности одного конкретного пользователя.

Варианты использования могут быть полезны как на этапе моделирования деловой активности, так и при описании внутреннего поведения информационных систем. Одна из опасностей заключается в смешении двух уровней. Лучше этого не делать, поскольку они представляют интерес для различных аудиторий.

Представленный в виде варианта использования диалог пользовательского интерфейса описывает, какой информацией пользователь обмениваются с системой, а не то, как она представлена на экране. Это естественным образом приводит к реализации XML, в которой информационное содержание отделено от особенностей представления.

Диаграммы взаимодействия объектов

Диаграммы взаимодействия объектов позволяют проанализировать обмен сообщениями между объектами на более тонком уровне детализации, чем модель потока данных.

Диаграммы взаимодействия объектов неоценимы, если требуется описать взаимодействие между различными системами. Они позволяют определить, какая информация содержится в каком сообщении. Поскольку эти сообщение написаны на языке XML, диаграммы взаимодействия объектов дают нам контекст, требуемый для начала проектирования структуры XML каждого индивидуального сообщения.

Пример статической информационной модели

Информационную модель данных, которая затем будет трансформирована в XML-документ, можно представить в виде *древовидной* структуры, которая начинается с *корня* и заканчивается *листьями*.

XML-документ должен содержать *корневой элемент*, который является *родительским* для всех остальных элементов. Любой элемент (кроме находящихся на самом нижнем уровне дерева) может иметь вложенные элементы (*дочерние элементы*).

С помощью терминов *родитель*, *дочерний* и *потомок* описываются отношения между элементами в дереве XML документа. *Родители* содержат *дочерние элементы*, а дочерние элементы одного уровня называются *потомками* (*братьями* или *сестрами*).

На рисунке представлен пример статической информационной модели, описывающей отношения между элементами в предметной области «Книжный магазин».

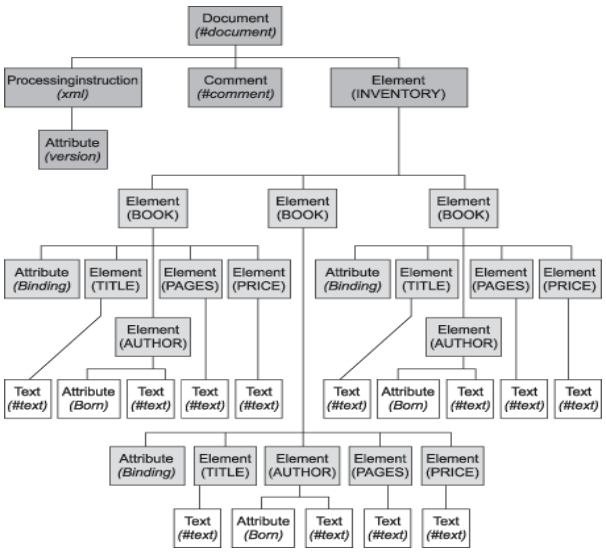


РИСУНОК 1

```
<?xml version="1.0" encoding="windows-1251" ?>

<!-- Имя файла: Inventory Dom.xml -->
<INVENTORY>
  <BOOK Binding="mass market paperback">
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR Born="1835">Mark Twain</AUTHOR>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
```

```
  <BOOK Binding="trade paperback">
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR Born="1804">Nathaniel Hawthorne</AUTHOR>
    <PAGES>473</PAGES>
    <PRICE>$10.95</PRICE>
  </BOOK>
  <BOOK Binding="hardcover">
    <TITLE>Moby-Dick</TITLE>
    <AUTHOR Born="1819">Herman Melville</AUTHOR>
    <PAGES>724</PAGES>
    <PRICE>$9.95</PRICE>
  </BOOK>
</INVENTORY>
```

Методические указания ко второй части лабораторной работы

Синтаксис XML

Для ограничения тегов в разметке XML, так же как и в HTML, используются угловые скобки: тег начинается со знака "меньше" (<) и завершается знаком "больше" (>). Но необходимо помнить, что в отличие от HTML вся разметка XML чувствительна к регистру символов, это касается как имен тегов, так и значений атрибутов.

Имена

В языке XML все имена должны начинаться с буквы, символа нижнего подчеркивания (_) или двоеточия (:) и продолжаться только допустимыми для имен символами, а именно: они могут содержать только буквы, входящие в секцию букв кодировки Unicode, арабские цифры, дефисы, знаки подчеркивания, точки и двоеточия. Однако имена не могут начинаться со строки *xml* в любом регистре. Имена, начинающиеся с этих символов, зарезервированы для использования консорциумом W3C.

Структура XML- документа

Любой XML-документ состоит из следующий частей:

- Необязательный пролог
- Тело документа
- Необязательный эпилог, следующий за деревом элементов

Рассмотрим каждую из частей более подробно.

Пролог

Пролог состоит из нескольких частей:

Необязательное *объявление XML* (XML Declaration).

Объявление заключено между символами <?...?> и содержит:

пометку xml и номер версии (*version*) спецификации XML;

указание на кодировку символов (*encoding*), в которой написан документ (по умолчанию encoding="UTF-8");

параметр standalone, который может принимать значения "yes" или "no" (по умолчанию standalone="yes"). Значение "yes" показывает, что в документе

содержатся все требуемые декларации элементов, а "no" - что нужны внешние определения DTD.

Все это вместе может выглядеть следующим образом:

<?xml version ="1.0" encoding-"windows-1251" standalone="yes"?>

Важно отметить, что в объявлении XML только атрибут *version* является обязательным, все остальные атрибуты могут быть опущены и, следовательно, принимать значения по умолчанию. Также нужно помнить, что все эти атрибуты следует указывать только в приведенном выше порядке.

Комментарии

Инструкции по обработке.

Назначение инструкций по обработке — сообщить информацию, передаваемую XML-процессором приложению. Инструкция по обработке имеет следующую общую форму записи:

<? Кому инструкция ?>

Здесь *Кому* есть имя приложения, которому адресована инструкция. Допускается любое имя при соблюдении следующих правил:

имя должно начинаться с буквы или символа подчеркивания (_), после чего могут следовать или не следовать другие буквы, цифры, точки (.), тире (-) или символы подчеркивания (_);

имя «xml», в любом сочетании строчных или прописных букв, зарезервировано («xml» строчными буквами используется в объявлении XML-документа, которое представляет собой разновидность инструкции по обработке).

Инструкция есть информация, передаваемая приложению. Она может состоять из любой последовательности символов, за исключением пары ?>, зарезервированной для обозначения окончания инструкции по обработке.

Например, следующая инструкция по обработке предписывает браузеру использовать CSS-таблицу из файла styles.css:

<?xml-stylesheet type="text/css" href=" styles.css"?>

Символы пустых пространств.

Необязательное *объявление типа документа*, DTD (Document Type Declaration), которое заключено между символами <!DOCTYPE...> и может занимать несколько строк. В этой части объявляются теги, использованные в документе, или приводится ссылка на файл, в котором записаны такие объявления.

После объявления типа документа также могут следовать комментарии, команды обработки и символы пустых пространств.

Поскольку все эти части необязательны, пролог может быть опущен.

Тело документа

Тело документа состоит из одного или больше *элементов*. В правильно оформленном XML-документе элементы формируют простое иерархическое дерево, в котором обязательно присутствует корневой элемент (root element), в который вложены все остальные элементы документа. Имена элементов должны быть уникальны в пределах документа. Имя корневого элемента считается именем всего документа и указывается во второй части пролога после слова Doctype.

Элемент начинается открывающим тегом, затем идет необязательное содержимое элемента, после чего записывается закрывающий тег (в отличие от HTML, наличие закрывающего тега обязательно, исключением являются элементы

без содержания, так называемые *пустые элементы*, которые могут быть записаны в сокращенной форме: <имя_элемента/>). В качестве содержимого элемента могут выступать:

Другие элементы

Символьные данные

Ссылки на символы

Для того чтобы вставить в текст документа некоторый символ, который, например не присутствует в раскладке клавиатуры либо может быть неправильно истолкован анализатором, используют ссылки на символы. Ссылка на символ обязательно начинается со знака "&" (амперсанта) и заканчивается точкой с запятой. Ссылки на символы записываются в следующем виде:

&# код_символа_в_Unicode;

Код символа можно записать и в шестнадцатеричном виде. В этом случае перед ним ставится символ "x":

&#xШестнадцатеричный_код_символа;

Ссылки на сущности

Ссылки на сущности позволяют включать любые строковые константы в содержание элементов или значение атрибутов. Ссылки на сущности, как и ссылки на символы, начинающиеся с амперсанта, после которого идет имя сущности и заканчивающиеся точкой с запятой:

&имя_сущности;

Ссылки на сущности указывают программе-анализатору подставить вместо них строку символов, заранее заданную в определении типа документа (DTD).

Комментарии

Если надо вставить в текст документа комментарий либо сделать какой-то фрагмент "невидимым" для программы-анализатора, то его оформляют следующим образом:

<!--…текст комментария…-->

Разделы **CDATA**

Секция CDATA используется, для того чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы. Программа-анализатор не разбивает секцию CDATA на элементы, а считает ее просто набором символов. В отличие от комментариев, содержание данной секции не игнорируется, а передается без изменений на выход программы анализатора, благодаря чему его можно использовать в приложении.

Секция CDATA начинается со строки <![CDATA[, после которой записывается содержимое секции. Завершается секция двумя закрывающими квадратными скобками и знаком "меньше":

<![CDATA[содержание секции]]>

Инструкции по обработке

Инструкции по обработке содержат указания программе-анализатору документа XML. Инструкции по обработке заключаются между символами <? и ?>. Сразу за начальным вопросительным знаком записывается имя программного

модуля, которому предназначена инструкция. Затем, через пробел, идет сама инструкция, передаваемая программному модулю. Сама инструкция это обычная строка, которая не должна содержать набор символов "?>", означающий конец инструкции. Примером инструкции по обработке может служить строка объявления XML:

<?xml version="1.0" encoding="windows-1251"?>

Эта инструкция предназначена программе, обрабатывающей документ XML. Инструкция передает ей номер версии и кодировку, в которой записан документ.

Атрибуты

Открывающие теги либо теги пустых элементов в XML могут содержать атрибуты, представляющие собой пару *имя=значение*. В одном открывающем теге разрешается использовать только один экземпляр имени атрибута. Атрибуты могут содержать ссылки на объекты, ссылки на символы, текстовые символы. В отличие от языка HTML, в XML значения атрибутов обязательно надо заключать в апострофы ('), либо в кавычки ("). Таким образом, атрибут может быть записан в одном из двух форматов:

имя_атрибута="значение_атрибута"

имя_атрибута='значение_атрибута'

Атрибуты используются для того, чтобы связать некоторую информацию с элементом, а не просто включить ее в содержание последнего. Однозначного ответа на вопрос «Что лучше выбрать – элемент или атрибут?» не существует. Каждый выбирает то, что ему больше нравится. Атрибуты удобно использовать для описания простых значений или для указания типа элемента. Например, мы можем ввести в открывающий тег <city> атрибут type (который может принимать одно из значений: город, поселок, деревня). Тогда данный тег может выглядеть следующим образом:

<city type="город"> Новосибирск </city>

Эпилог

В эпилог XML могут входить комментарии, инструкции по обработке и/или пустое пространство.

В языке XML не определен индикатор конца документа, большинство приложений для этой цели используют завершающий тег корневого элемента документа. Таким образом, встретив завершающий тег корневого элемента, скорее всего, приложение закончит обработку документа, и эпилог обработан не будет.

Правильно оформленные и валидные документы

В общем случае XML- документы должны удовлетворять следующим требованиям:

Любой XML-документ должен всегда начинаться с инструкции <?xml?>, внутри которой также можно задавать номер версии языка, номер кодовой страницы и другие параметры, необходимые программе-анализатору в процессе разбора документа.

Теги (метаданные) в документе выделяются символами <> (угловые скобки). Название тега должно начинаться сразу после угловой скобки (пробелы между открывающей угловой скобкой и названием тега недопустимы). Каждый

открывающий тэг, определяющий некоторую область данных в документе, обязательно должен иметь своего закрывающего «напарника», нельзя опускать закрывающие тэги.

В XML учитывается регистр символов.

Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки.

Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов.

Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные, и поэтому учитываются все символы форматирования (т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Если XML-документ не нарушает приведенные правила, то он называется *формально-правильным* и все анализаторы, предназначенные для разбора XML-документов, смогут работать с ним корректно.

Пример XML-документа

Ниже представлена структура XML-документа по приведенной в первой части данной лабораторной работы статической информационной модели предметной области «Расписание занятий».

```
<?xml version="1.0"?>
<timetable>
    <day dayOfWeek="Monday">
        <lesson type="practical">
            <timeFrom>08.00</timeFrom>
            <timeTo>09.30</timeTo>
            <subject>Deutsch</subject>
            <teacher>Borisova</teacher>
            <room>216</room>
        </lesson>
        <lesson type="lecture">
            <timeFrom>09.40</timeFrom>
            <timeTo>11.10</timeTo>
            <subject>SAP Administration</subject>
            <teacher>Egorov</teacher>
            <room>384</room>
        </lesson>
        <lesson  type="practical">
```

```
            <timeFrom>11.20</timeFrom>
            <timeTo>12.50</timeTo>
            <subject>SAP Administration</subject>
            <teacher>Petrov</teacher>
            <room>384</room>
        </lesson>
    </day>
    </timetable>
```

Прежде чем Internet Explorer отобразит XML-документ, его встроенный синтаксический XML-анализатор (parser) просмотрит содержимое документа. Если он обнаружит ошибку, Internet Explorer отобразит страницу с сообщением об ошибке, не предпринимая попытки отобразить документ.

Internet Explorer использует для отображения документа имеющуюся по умолчанию таблицу стилей. Именно поэтому в описании ошибки упоминается использование таблицы стилей XSL.

Лабораторная работа № 2 . Создание динамических web - страниц на основе языков xml, xsl

Цель работы:

Отображение XML-документа в виде HTML-страницы. Применение таблиц стилей. Вставка элементов HTML в XML-документы

Результат:

Динамическая страница на основе XML файла.

Этапы выполнения работы

Методы создания XSL приложений

В XML-документе, созданном в первой лабораторной работе, предусмотрите возможность хранения URL для создания гиперссылок и URL графических файлов.

Выполните отображение этого XML-документа с помощью простой и вложенной таблицы.

Добавьте постраничное отображение.

Выполните отображение фрагмента этого же XML-документа с помощью сцепления не табличных элементов HTML с XML-элементами.

Прежде чем приступить к выполнению лабораторной работы, выполните следующие практические работы.

ПРАКТИЧЕСКАЯ РАБОТА № 1. Стилевые таблицы XSL, JavaScript и XML

РАБОТАЕМ ВМЕСТЕ:

Рассмотрим простой пример XML-файла (ex01.xml).

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<tutorial>
<title>&amp;quot;Путешествие дилетантов&amp;quot;</title>
<author>Булат Окуджава</author>
</tutorial>
```

Если мы откроем этот файл в браузере Internet Explorer, то мы увидим тот же самый текст, который приведен выше, вместе со всеми тегами и служебной информацией.

Нам не нужны теги и служебная информация! Мы хотим видеть только ту информацию, которая относится к делу, а при помощи тегов - управлять внешним видом этой информации. Эта задача решается легко и просто: необходимо к XML-файлу добавить шаблон преобразования - XSL-файл.

Перепишем наш XML-файл в следующем виде (ex01-1.xml).

```
<?xml version="1.0" encoding="WINDOWS-1251" ?>
<?xml-stylesheet type='text/xsl' href="ex01-1.xsl" tppabs="ex01-1.xsl"?>
<NewDataSet>
    <tutorial>
        <title>Путешествие дилетантов</title>
        <author>Булат Окуджава</author>
    </tutorial>
</NewDataSet>
```

И создадим XSL-файл ex01-1.xsl. Текст файла приведен ниже.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template  match="/">
<p><strong><xsl:value-of  select="//title"/></strong></p>
<p><xsl:value-of  select="//author"/></p>
</xsl:template>
</xsl:stylesheet>
```

Если мы теперь откроем файл ex01-1.xsl в браузере Internet Explorer, то мы увидим, что наша задача решена, - на экране осталась только необходимая нам информация, все теги исчезли. Результат, который вы получите на экране браузера, приведен ниже.

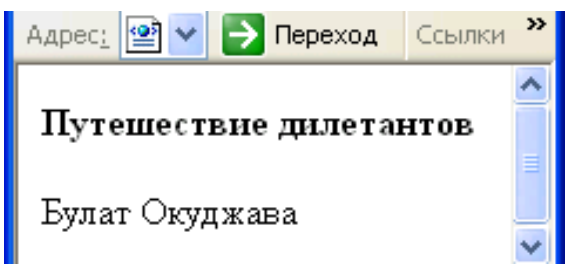


РИСУНОК 2

Легко также увидеть, что порядок вывода строк у нас определяется только содержанием шаблона преобразования - XSL-файла. При необходимости шаблон можно легко поменять, абсолютно не меняя наш основной XML-файл.

Перепишем XML-файл. Информационную часть изменять не будем, а шаблон укажем другой ex01-2.xml.

```
<?xml version="1.0" encoding="WINDOWS-1251" ?>
<?xml-stylesheet type='text/xsl' href="ex01-2.xsl" tppabs="ex01-2.xsl"?>
<tutorial>
        <title>Путешествие дилетантов</title>
        <author>Булат Окуджава</author>

</tutorial>
```

Создадим XSL-файл ex01-2.xsl. Текст файла приведен ниже.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template  match="/">
<p><xsl:value-of  select="//author"/></p>
<p><strong><xsl:value-of  select="//title"/></strong></p>
</xsl:template>

</xsl:stylesheet>
```

Если мы теперь откроем файл ex01-2.xsl в браузере Internet Explorer, то результат будет другим.

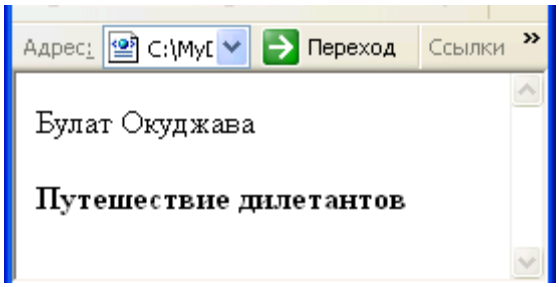


РИСУНОК 3

Отметим теперь момент, который является ключевым для разработчиков баз данных. Информация в XML-странице появляется, как правило, в результате запроса к базе данных. Запрос к базе данных в многопользовательской среде - это весьма дорогостоящая операция. Предположим теперь, что у нас нет XML и мы формируем стандартные статические HTML-страницы. В этом случае для решения задачи простого преобразования внешнего представления информации, например, для изменения сортировки, у нас есть два способа решения проблемы: выполнить запрос и сохранить результаты в каком-либо временном буфере на сервере или каждый раз при изменении внешнего представления выполнять новый запрос и формировать HTML-страницу заново.

Первый способ требует трудоемкого программирования, второй способ значительно увеличивает нагрузку на сервер базы данных, производительность которого часто является узким местом системы, - пользователю всегда хочется получать результаты быстрее.

XML и XSL - это исчерпывающее решение описанной выше проблемы. Фактически XML-страница - это и есть временный буфер для результатов запросов. Только вместо нестандартного и трудоемкого программирования мы теперь используем стандартный механизм XSL.

Есть и еще одно соображение, которое может быть существенным для разработчиков баз данных. Большинство современных СУБД могут форматировать результаты запроса к базе данных в виде XML-файла. То есть при построении интерфейса пользователя в рамках технологии XML и XSL мы добиваемся определенной независимости от поставщика СУБД. В части организации вывода - практически полной независимости. А эта часть весьма велика в большинстве прикладных систем, ориентированных на работу с базами данных. Конечно, помимо вывода есть еще ввод и серверная обработка бизнес-логики, но здесь вам придется искать какие-то иные решения.

Разберем теперь более подробно первый пример. Напомним его текст.

Первая строка информирует браузер о том, что файл имеет формат XML. Атрибут version является обязательным. Атрибут encoding не является обязательным, но если у вас в тексте есть русские буквы, то необходимо вставить этот атрибут, в противном случае XML-файл просто не будет обрабатываться, - вы получите сообщение об ошибке.

Следующие строки - это тело XML-файла. Оно состоит из элементов, которые в совокупности образуют древовидную структуру. Элементы идентифицируются тегами и могут быть вложены друг в друга.

Элементы могут иметь атрибуты, значения которых тоже могут обрабатываться в соответствии с шаблоном.

На верхнем уровне XML-файла всегда находится один элемент. То есть файл вида

```
<?xml version="1.0" encoding="WINDOWS-1251" ?>
<?xml-stylesheet type='text/xsl' href="ex01-1.xsl" tppabs="ex01-1.xsl"?>
<NewDataSet>
	<tutorial>
		<title>Путешествие дилетантов</title>
		<author>Булат Окуджава</author>
	</tutorial>
	<tutorial>
		<title>Привет школяр!</title>
		<author>Булат Окуджава</author>
	</tutorial>
	<tutorial>
		<title>Четки</title>
		<author>Анна Ахматова</author>
	</tutorial>
</NewDataSet>
```

не будет обрабатываться браузером. Для преобразования в корректный XML-файл нужно добавить теги элемента верхнего уровня, например

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<knowledgeDatabase>
<tutorial>
        <title>Путешествие дилетантов</title>
        <author>Булат Окуджава</author>
    </tutorial>
    <tutorial>
        <title>Привет школяр!</title>
        <author>Булат Окуджава</author>
    </tutorial>
    <tutorial>
        <title>Четки</title>
        <author>Анна Ахматова</author>
    </tutorial>
</knowledgeDatabase>
```

Перейдем теперь к шаблону преобразования - к XSL-файлу. Задача XSL-файла - преобразовать дерево XML-файла в другое дерево, которое, например, будет соответствовать формату HTML и может быть изображено на экране браузера с учетом форматирования, выбора шрифтов и т.п.

Для того, чтобы браузер выполнил необходимое преобразование, нужно в XML-файле указать ссылку на XSL-файл

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<?xml-stylesheet type='text/xsl' href='ex01-1.xsl'?>
```

Рассмотрим теперь текст XSL-файла

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<p><strong><xsl:value-of select="//title""/></strong></p>
<p><xsl:value-of select="//author"/></p>
</xsl:template>
</xsl:stylesheet>
```

Первая строка файла содержит тег элемента xsl:stylesheet. Атрибуты элемента — номер версии и ссылка на пространство имен. Эти атрибуты элемента xsl:stylesheet являются обязательными. В нашем случае пространство имен - это все имена элементов и их атрибутов, которые могут использоваться в XSL-файле. Для XSL-файлов ссылка на пространство имен является стандартной.

Заметим, что XSL-файл является одной из разновидностей XML-файлов. Он не содержит пользовательских данных, но формат его тот же самый. Файл содержит элемент верхнего уровня xsl:stylesheet, а далее идет дерево правил преобразования.

В настоящем документе мы не будем подробно пояснять, что означает каждый элемент XSL-файла. Мы будем приводить различные примеры и показывать результат в каждом примере. Читатель сможет самостоятельно сопоставить различные элементы XSL-файла и инициируемые этими элементами преобразования исходного XML-файла с пользовательской информацией. Вы всегда

сможете открыть реальный файл и посмотреть все в цвете. При необходимости закомментируйте ссылку на XSL-файл. Синтаксис комментария следующий - <!-- Текст комментария -->. В текст комментария нельзя вставлять символы --.

В первом примере мы посмотрели, как с помощью элемента xsl:value-of можно вывести в HTML-формате содержание элемента (текст, заключенный между тегами). Теперь мы посмотрим, как при помощи того же самого элемента можно вывести значение атрибута элемента.

Рассмотрим следующий XML-файл ex02-1.xml

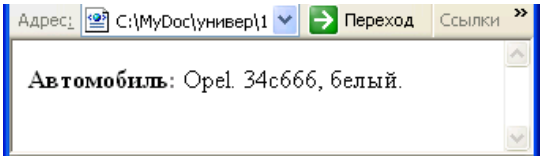
```
<?xml version="1.0" encoding="WINDOWS-1251" ?>
<?xml-stylesheet type='text/xsl' href="ex02-2.xsl" tppabs="ex02-2.xsl"?>
<tutorial>
        <car caption="Автомобиль: " marka="Opel">
                <carInfo number="34c666" color="белый" />
        </car>

</tutorial>
```

В этом файле информация хранится не в содержании элементов, а в виде значений атрибутов. Файл ex02-1.xsl имеет вид

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template  match="/">
<P><B><xsl:value-of select="//car/@caption"/></B>
<xsl:value-of select="//car/@marka"/>.
<xsl:value-of select="//car/carInfo/@number"/>,
<xsl:value-of select="//carInfo/@color"/>.</P>
</xsl:template>
</xsl:stylesheet>
```

Обратите внимание на синтаксис ссылки на атрибут элемента - //car/**@**Marka. Имя элемента и имя атрибута разделены парой символов "/@". В остальном синтаксис тот же самый, что и для ссылки на содержание элемента.

Результат имеет следующий вид:



Обратим теперь внимание на следующий момент. В XSL-файле мы никак не использовали элемент tutorial. На самом деле можно было использовать полный путь. Перепишем наш XML-файл, увеличив глубину дерева (ex02-2.xml)

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<?xml-stylesheet type='text/xsl' href="ex02-2.xsl" tppabs="ex02-2.xsl"?>
<tutorial>
<avto>
        <car caption="Автомобиль: " marka="Opel">
```

```
            <carInfo number="34c666" color="белый" />
        </car>
</avto>

</tutorial>
```

Файл ex02-2.xsl имеет вид

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template  match="/">
<P><B><xsl:value-of select="//avto/car/@caption"/></B>
<xsl:value-of select="//avto/car/@marka"/>.
<xsl:value-of select="//avto/car/carInfo/@number"/>, <xsl:value-of
select="//carInfo/@color"/>.</P>
</xsl:template>
</xsl:stylesheet>
```

Результат будет тем же самым.

В этом примере мы использовали полную ссылку для значений атрибутов. При выводе одиночных значений оба варианта - полная и сокращенная ссылка - работают одинаково.

На этом мы закончим разбор примеров с выводом одиночных значений и перейдем к выводу табличной информации - к выводу результатов запроса.

Вывод результатов запроса

До тех пор, пока мы работаем с несколькими реквизитами одного и того же объекта, разницы между XML и HTML практически нет. Однако стоит нам перейти к информации, содержащей несколько строк, как выгоды XML становятся очевидны. Но прежде чем перейти к выгодам, научимся выводить на экран простую таблицу.

Рассмотрим следующий XML-файл - ex03.xml. Текст его приведен ниже.

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<?xml-stylesheet type='text/xsl' href="ex03-1.xsl" tppabs="ex03-1.xsl"?>
<tutorial>
<avto>
 <cars>
   <carsCaption>Автомобили</carsCaption>
   <carsCaptionMarka>Марка</carsCaptionMarka>
   <carsCaptionNumber>Номер</carsCaptionNumber>
   <carsCaptionColor>Цвет</carsCaptionColor>
   <car>
     <carMarka>Opel</carMarka>
     <carNumber>34c666</carNumber>
     <carColor>Белый</carColor>
   </car>
    <car>
     <carMarka>BMW</carMarka>
     <carNumber>34rt66</carNumber>
```

```
    <carColor>Индиго</carColor>
   </car>
   <car>
    <carMarka>Mersedes</carMarka>
    <carNumber>341111</carNumber>
    <carColor>черный</carColor>
   </car>
   <car>
    <carMarka>Mazda</carMarka>
    <carNumber>456678</carNumber>
    <carColor>синий</carColor>
   </car>
  </cars>
</avto>

</tutorial>
```

Предположим, что это результат запроса к базе данных и выведем на экран соответствующую таблицу.

Простая таблица

Первый шаг - это, как всегда, добавление шаблона преобразования. Модифицируем наш файл, добавив в него ссылку на шаблон. В результате получим файл ex03-1.xml.

В этот файл добавлен шаблон преобразования ex03-1.xsl.

Рассмотрим этот шаблон подробнее. Вот его текст.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<table border="1" >
  <tr bgcolor="#C1FFC1">
    <td align="center"><strong><xsl:value-of select="//carsCaptionMarka"/>
</strong></td>
    <td align="center"><strong><xsl:value-of select="//carsCaptionNumber"/>
</strong></td>
    <td align="center"><strong><xsl:value-of select="//carsCaptionColor"/>
</strong></td>
  </tr>
  <xsl:for-each select="tutorial/avto/cars/car">
  <tr bgcolor="#11dd33">
    <td><xsl:value-of select="carMarka"/></td>
    <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
    <td><xsl:value-of select="carColor"/></td>
  </tr>
  </xsl:for-each>
</table>
</xsl:template>
```

</xsl:stylesheet>

Первые две строки шаблона являются уже привычными. Следующие шесть строк — это строка, содержащая заголовки столбцов таблицы. Конструкция для извлечения текста заголовков таблицы вам уже знакома. А вот девятая строка является новой:

<xsl:for-each select="tutorial/avto/cars/car">

Этот элемент шаблона позволяет выбрать и просмотреть все группы информации, полный путь к которым задается списком тегов " tutorial/avto/cars/car ". Обратите внимание - путь задается полностью, ни один из тегов опустить нельзя. Далее в ячейки таблицы помещается информация о наших автомобилях. В отличие от первых примеров путь к соответствующей информации тоже задается полностью. Попробуем, например, разместить информацию о марке чуть-чуть иначе ex03-2.xml:

<carMarka>
<carNick>Opel</carNick>
</carMarka>

Если мы в соответствующем XSL-файле поставим ссылку <xsl:value-of select="carNick"/>, то в соответствующем столбце никакойй марки мы не увидим. Ссылка должна быть полной - <xsl:value-of select=" carMarka / carNick "/>. Правильный результат приведен ниже.

Марка	Номер	Цвет
Мерседес	341111	Черный
Шкода	456678	Красный
Опель	34c666	Белый
БМВ	342rt66	Индиго

Марка	Номер	Цвет
Opel	34c666	Белый
BMW	34rt66	Индиго
Mersedes	341111	черный
Mazda	456678	синий

Сортировка в XSLT применима к некоторому множеству узлов. Она изменяет типичную для XSLT последовательность вывода элементов (в порядке просмотра докумнента) и определяется наличием в элементе **xsl:apply-templates** одного или нескольких элементов **xsl:sort**. Синтаксис этого элемента прост:

```
<xsl:sort
   select="выражение"
   datatype="тип данных"
   order={"ascending" | "descending"}
   case-order={"upper-first" | "lower-first"} />
```

Атрибут **select** содержит выражение, оцениваемое для каждого узла множества и являющееся критерием сортировки. Атрибут **order** определяет порядок

сортировки: значение "ascending" ("восходящий") означает сортировку по возрастанию и является **значением по умолчанию**, значение "descending" ("нисходящий") означает сортировку по убыванию. Если необходимо провести сортировку по нескольким критериям (например по фамилии и по имени), то в этом случае первым записывается элемент **xsl:sort**, выполняющий сортировку по фамилии, затем элемент **xsl:sort**, выполняющий сортировку по имени. Давайте рассмотрим все на простом примере: сортировка книжного каталога:

Входящий документ

```
<catalogtitle="Библиотека web-мастера">
   <book title="XHTML 1.0" category="Языки разметки" author="Иванов И.Г."/>
   <book title="Javascript для профессионалов" category="Языки скриптов"
author="Семенов В.У."/>
   <book title="Apache" category="Web-серверы" author="Фримен Г."/>
   ...
</catalog>
```

Преобразование

```
<xsl:template match="/">
   <body>
      <table border="1" width="80%" alibn="center">
         <xsl:apply-templates select="catalog/book">
            <xsl:sort select="@category" />
         </xsl:apply-templates>
      </table>
   </body>
</xsl:template>
<xsl:template match="book">
   <tr>
      <td><xsl:value-of select="@title" /></td>
      <td><xsl:value-of select="@author" /></td>
      <td><xsl:value-of select="@category" /></td>
   </tr>
</xsl:template>
```

Пример, который вы видите является несколько более "навороченным", чем приведенный в уроке, однако это не меняет сути - присутствие элемента **xsl:sort** в теле элемента **xsl:apply-templates** заставляет выбранное множество узлов сортироваться по значению выражения, указанного атрибутом **select** элемента **xsl:sort**. Очевидно, что сортировка происходит в алфавитном порядке по возрастанию значений выражения **select**.

Давайте теперь изменим направление сортировки - сделаем сортировку по убыванию значений выражения. Для этого изменим приведенный выше код следующим образом: добавим в элемент **xsl:sort** атрибут **order** со значением "descending". Вот так:

```
   ...
   <xsl:sort order="descending" />
   ...
```

Видите, что направление сортировки поменялось.

Но давайте рассмотрим такой пример: книги в каталоге имеют стоимость, по которой и необходимо осуществлять сортировку. Например такие значения стоимостей книг, как "3.1", "20.5" бубут отсортированны по возрастанию так: "20.5", "3.1". Как упоминалось ранее, **xsl:sort** выполняет алфавитную (строковую) сортировку, а символ "3" больше символа "2", а значит "20.5" меньше "3.1". С точки зрения нашей логики получится абсолютный бред! Как же решить эту проблему? Решение кроется в использовании **типа данных**, указываемого атрибутом **data-type**. Если **data-type**="text" (по умолчанию), то происходит строковая сортировка; при **data-type**="number" сортировка числовая (то что нам надо):

```
...
<xsl:sort select="@price" data-type="number" />
...
```

Последний атрибут **case-order** определяет порядок сортировки по регистру символов, что может быть актуально при строковых сортировках. Значение "upper-first" (верхний регистр вперед) означает, что символы в в верхнем регистре будут рассмотрены и отсортированны раньше, чем символы нижнего регистра. Значение "lower-first" - наоборот: символы нижнего регистра рассматриваются и сортируются раньше. Например, при **case-order**="lower-first" строка **"Пример"** помещается перед строкой **"пример"**.

В предыдущих примерах порядок строк в таблице полностью соответствовал группам тегов в XML-файле. Этот порядок можно изменять. Добавим в тег

<xsl:for-each select="tutorial/avto/cars/car">

атрибут order-by

<xsl:for-each select="tutorial/avto/cars/car" order-by="carMarka">

Наша таблица примет вид (ex03-3.xml, ex03-3.xsl).

Марка	Номер	Цвет
БМВ	342rt66	Индиго
Мерседес	341111	Черный
Опель	34c666	Белый
Шкода	456678	Красный

Марка	Номер	Цвет
Опель	1569875	Белый
Мерседес	341111	Черный
Шкода	456678	Красный
БМВ	546900	Индиго

Более интересные результаты мы получим, если попытаемся отсортировать таблицу по столбцу "Number". Вначале попробуем сделать по аналогии с предыдущим примером - атрибут order-by="carMarka" заменим на order-by="carNumber". Результат приведен ниже (ex03-4.xml, ex03-4.xsl).

Таблица действительно отсортирована по столбцу "номер", но это не числовая, а строковая сортировка! Для того, чтобы браузер воспринял значения как числа, ему необходимо об этом сказать, - вместо order-by="carNumber". необходимо написать order-by="number(carNumber)". Теперь мы получили правильный результат (ex03-5.xml, ex03-5.xsl).

Приведем теперь пример сортировки по нескольким столбцам. Различные элементы в атрибуте order-by должны разделяться символом ";" - order-by=" carMarka ;number(carNumber) " (ex03-6.xml, ex03-6.xsl). Таблица приведена ниже.

Марка	Номер	Цвет
Мерседес	341111	Черный
		Красный
		Индиго
		Белый

Марка	Номер	Цвет
БМВ	246900	Белый
БМВ	546900	Индиго
БМВ	746900	Морковный
Мерседес	541111	Зеленый
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
Мерседес	941111	Черный
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый
Шкода	156678	Черная
Шкода	456678	Красный

Следующий пример работает только под управлением XML-парсера версии 3. В нем строки сортируются по одному столбцу - по марке автомобиля. Этот пример уже приводился выше, однако теперь мы используем новый синтаксис (ex03-7.xml, ex03-7.xsl).

Отметим разницу.

При использовании нового синтаксиса используется ссылка на другое пространство имен

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Это очень важный момент, и его никогда нельзя упускать из виду.

Кроме того, мы убрали атрибут order-by в элементе xsl:for-each и добавили другой элемент

<xsl:sort order="ascending" select="carMarka"/>

Если элемент xsl:sort присутствует в элементе xsl:for-each, то он всегда должен стоять сразу после элемента xsl:for-each. Синтаксис элемента xsl:sort достаточно очевиден. В нем используются два атрибута: атрибут order - способ сортировки (по возрастанию или по убыванию) и атрибут select - имя поля, по которому производится сортировка. Если нам нужно отсортировать по первому элементу, как в данном примере, то вместо " carMarka " можно было поставить точку - ".", для других элементов нужно указывать его имя, например "carColor", если нам нужно отсортировать записи по цвету машины. На самом деле атрибутов может быть пять - select, lang, data-type, order и case-order, но мы не будем здесь

рассматривать все эти атрибуты, поскольку здесь мы не преследуем цель дать полное описание всех элементов, используемых в XSL, и их атрибутов.

Таблица результатов приведена ниже.

Марка	Номер	Цвет
БМВ	546900	Индиго
БМВ	746900	Морковный
БМВ	246900	Белый
Мерседес	941111	Черный
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
Мерседес	541111	Зеленый
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый
Шкода	456678	Красный
Шкода	156678	Черная

С использованием нового синтаксиса легко сменить сортировку по возрастанию на сортировку по убыванию (ex03-8.xml, ex03-8.xsl). Этот пример работает только под управлением XML-парсера версии 3.

Разница заключается в одной строке

<xsl:sort order="descending" select="carMarka"/>

Мы изменили значение атрибут order - значение ascending заменено на descending.

Таблица результатов приведена ниже.

Марка	Номер	Цвет
Шкода	156678	Черная
Шкода	456678	Красный
Опель	7569875	Фиолетовый
Опель	2569875	Красный
Опель	1569875	Белый
Мерседес	541111	Зеленый
Мерседес	741111	Перломутровый
Мерседес	641111	Оливковый
Мерседес	941111	Черный
БМВ	246900	Белый
БМВ	746900	Морковный
БМВ	546900	Индиго

Покажем теперь сортировку по нескольким полям (ex03-9.xml, ex03-9.xsl). Этот пример работает только под управлением XML-парсера версии 3.

В этом примере у нас фигурируют две строки с элементом

<xsl:sort. <xsl:sort order="ascending" select="carMarka"/>

<xsl:sort order="ascending" select="number(carNumber)" data-type="number"/>

Строки вначале сортируются по марке автомобиля, а затем по их номерам. Обратите внимание - для того, чтобы сортировка выполнялась в числовой последовательности, в элемент xsl:sort мы добавили атрибут data-type. Таблица результатов приведена ниже.

Марка	Номер	Цвет
БМВ	246900	Белый
БМВ	546900	Индиго
БМВ	746900	Морковный
Мерседес	541111	Зеленый
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
Мерседес	941111	Черный
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый
Шкода	156678	Черная
Шкода	456678	Красный

Элемент XSL:IF - фильтр

Рассмотрим теперь способы фильтрации строк таблицы. Первый пример использует старый синтаксис. В нем условие фильтрации указывается непосредственно в атрибуте select (ex04-1.xml, ex04-1.xsl).

Ниже приведена строка, в которую мы внесли необходимые изменения.

```
<xsl:for-each select="tutorial/avto/cars/car[carNumber$gt$1000000]" order-by="number(carNumber); carMarka">
```

И таблица результатов.

Марка	Номер	Цвет
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый

Вы видите, что в таблице остались только те автомобили, чей номер превышает 1000000, причем первым стоит Опель, чей номер меньше.

Все дальнейшие примеры в этом параграфе работают только под управлением XML-парсера версии 3.

Более гибкие возможности нам предоставляет новый синтаксис (ex04-2.xml, ex04-2.xsl). Обратите внимание - в новом синтаксисе атрибут order-by в элементе xsl:for-each не поддерживается, вместо него мы вставили два элемента xsl:sort.

```
<xsl:for-each select="tutorial/avto/cars/car">
  <xsl:sort order="ascending" select="number(carNumber)" data-type="number"/>
  <xsl:sort order="ascending" select="carMarka"/>
```

```
<xsl:if test="carNumber&gt;1000000">

<tr bgcolor="#11dd33">
  <td><xsl:value-of select="carMarka"/></td>
  <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
  <td><xsl:value-of select="carColor"/></td>
 </tr>
 </xsl:if>

 </xsl:for-each>
```

Кроме того, условие фильтра у нас вынесено в отдельный элемент xsl:if.

<xsl:if test="carNumber>1000000">

Не забывайте указывать конечный тег элемента xsl:if.

В этом примере таблица результатов полностью аналогична предыдущей.

Марка	Номер	Цвет
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый

Полностью преимущества нового синтаксиса проявляются при использовании функций.

Рассмотрим следующий пример (ex04-3.xml, ex04-3.xsl). В этом примере используется функция position(), определяющая порядковый номер фрагмента в исходном XML-файле.

Соответствующий элемент xsl:if.

<xsl:if test="position()<3"> и <xsl:if test="position()>5">

Результат.

Марка	Номер	Цвет
Мерседес	941111	Черный
Мерседес	641111	Оливковый

Марка	Номер	Цвет
Шкода	156678	Черная
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый
БМВ	546900	Индиго
БМВ	746900	Морковный
БМВ	246900	Белый

Стилевые таблицы XSL, Переменные и параметры

Переменные и **параметры** в XSLT очень похожи между собой и сильно отличаются от переменных в традиционных языках программирования. Классическое определение переменной, характерное для большинства языков программирования звучит следующим образом:

- Переменная - это имя, присвоенное ячейке памяти или ячейкам, которые могут содержать представление конкретного типа данных
- Значение переменной может изменяться в процессе выполнения программы.

С точки зрения логической модели преобразования, такие технические детали, как адрес переменной в памяти, нас не интересуют, хотя он несомненно существует. Гораздо более удобным является другое определение:

- Переменная - это объект определенного типа, с которым связанно **имя**

По этому имени мы можем обращаться к объекту, использовать его значение и т.д. Иными словами, в XSLT под переменной понимается не более чем ассоциация между значением и именем, и если мы скажем, что переменная **x** имеет значение **5**, это будет означать, что имя **"x"** связанно с объектом численного типа, значение которого равно **5**.

Теперь, когда с понятием переменной в XSLT кое-что прояснилось, отметим следующий факт: **переменнве в XSLT не могут быть измененны!** Разработчикам, которые до сих пор не имели опыта логического программирования и использовали в своей пректике традиционные языки программирования, будет нелегко с этим смириться. То, что *переменные* не могут быть *измененны*, дискредитирует само название - *переменные*, ибо они уже более похожи на константы.

В XSLT объявление переменной есть создание ассоциации между объектом и именем.

Элемент xsl:variable

Синтаксис:

```
<xsl:variable name="имя" select="выражение">
   <!-- Содержимое: шаблон -->
</xsl:variable>
```

Элемент **xsl:variable** в XSLT используется для связи имени переменной, указанного атрибутом **name** со значением. Значение переменной может быть полученно вычислением выражения, заднного необязательным атрибутом **select** или выполнением шаблона в содержимом элемента. Использовать объявленную таким образом переменную можно, указывая перед именем символ $:

$имя

Переменные XSLT могут быть глобальными и локальными. Если элемент xsl:variable является элементом верхнего уровня, то переменная будет глобальной; если элемент xsl:variable объявлен внутри шаблонного правила - локальной.

Областью видимости глобальной переменной является все преобразование. То есть значение переменной, объявленной элементом верхнего уровня, может быть использовано в преобразовании где угодно. К такой переменной можно обращаться даже до ее объявления. Единственное ограничение: **переменная не должна объявляться через собственное значение**.

Локальную переменную можно использовать только после объявления и только в том же родительском элементе, которому принадлежит объявляющий элемент **xsl:variable**. Если существует одноименная глобальная переменноя, то

локальная переменная в своей области видимости перекрывает ее. Имена локальных переменных могут совпадать, если их области видимости не пересекаются.

Использование переменных

Как правило, первой реакцией на известие о том, что переменные в XSLT нельзя изменять является реплика: "Да зачем они вообще тогда нужны?". Претензия со стороны процедурного программирования вполне серьезнаа и обоснованна - изменяемые переменные являютсся неотъемлимой частью сложных многошаговых вычислений. Тем не менее, переменные, даже в таком виде, в каком они присутсвуют в XSLT, являются очень важным и полезным элементом языка. Ниже перечислены наиболее типичные случаи использования переменных:

Переменные могут содержать значения выражений, которые многократно используются в преобразовании. Это избавит процессор от необходимости пересчитывать выражение каждый раз по новому.

Переменной может присваиваться результат преобразования, что позволяет манипулировать уже сгенерированными частями документа.

Попробуем разобраться на примерах.

Случай 1. Первый случай использования совершенно очевиден: если в преобразовании многократно используется выражение, требующее серьезных вычислений или просто громоздкое для записи, переменная может быть использованна для хранения единожды вычисленного результата. Например, если мы множество раз обращаемся ко множеству ссылок данного документа посредством вычисления выражения вида

//a

Гораздо удобнее и экономней с точки зрения вычислительных ресурсов объявить переменную вида

```
<xsl:variable name="links" select="//a" />
```

и использовать ее в преобразовании как $links. Например:

```
...количество ссылок в документе: <xsl:value-of select="count($links)" />
```

Другим примером к этому же случаю может служить следующая ситуация: выражение просто для вычисления, но очень громоздко для записи. Гораздо приятней один раз вычислить значение этого выражения и присвоить его какой нибудь переменной, а затем использовать вышеописонную нотацию для обращения к нему. Например:

Объявление

```
<xsl:variable name="image_path" select="http://www.site.com/chapter-files/images">
```

Использование

```
<img>
   <xsl:attribute name="src">
      <xsl:value-of select="concat($image_path, '/picture1_1.gif')" />
   </xsl:attribute
</img>
```

Случай 2. Второй типовый случай использования переменных также заметно облегчает создание выходного дерева, делая этот процесс гибким и легко

конфигурируемым. Примером формирования HTML документа при помощи такого подхода может быть следующий шаблон:

```
<xsl:template match="/">
   <html>
      <xsl:copy-of select="$head" />
      <xsl:copy-of select="$body" />
   </html>
</xsl:template>
```

Достоинство такого подхода состоит в том, что переменные, содержащие фрагменты деревьев, как бы становятся модулями, из которых в итоге собирается документ.

Более практичным применением возможности переменных содержать фрагменты деревьев является условное присваивание переменной значения. Представим себе следующий алгоритм:

```
   если условие1    то
      присвоить переменной1 значение1
   иначе
      присвоить переменной1 значение2
```

В традиционном языке программирования с изменяемыми переменными (например C++ или Javascript) это выглядело бы так:

```
   переменная1=условие1?значение1:значение2;
```

или в чуть более широком варианте:

```
   if (условие1)       переменная1=значение1
   else
      переменная1=значение2
```

Однако если в XSLT написать что нибудь подобное

```
<xsl:choose>
   <xsl:when test="условие1">
      <xsl:variable name="переменная1" select="значение1" />
   </xsl:when>
   <xsl:otherwise>
      <xsl:variable name="переменная1" select="значение2" />
   </xsl:otherwise>
</xsl:choose>
```

то требуемого результата все равно не достигли бы по той причине, что переменная будет иметь в этом случае локальную область видимости, ограниченную элементом **xsl:when** или **xsl:otherwise** и их дочерними элементами. Правильный шаблон для решения этой задачи выглядит следующим образом:

```
<xsl:variable name="переменная1"
   <xsl:choose>
      <xsl:when test="условие1">
         <xsl:copy-of select="значение1" />
      </xsl:when>
      <xsl:otherwise>
         <xsl:copy-of select="значение2" />
      </xsl:otherwise>
   </xsl:choose>
```

```
</xsl:variable>
```

Конечно, это не то же самое - на самом деле мы получаем не само значение, а дерево, содержащее это значение, но для строковых и численных значений особой разницы нет: дерево будет вести себя так, как будто это число или строка (см. правила приведения типов в одном из предыдущих уроков).

Параметры

Параметры в XSLT практически полностью идентичны переменным. Они точно так же связывают с объектом имя, посредством которого можно к ним обращиться. Главным различием является то, что значение, данное параметру при инициализации, является всего лишь значением по умолчанию, которое может быть переопределенно при вызове.

Давайте немного изменим угол зрения и посмотрим на шаблоны не как на правила преобразования, а представим их себе как *функции*, каждая из которых преобразует входной узел и возвращает фрагмент дерева в качестве результата. С этой точки зрения параметры шаблонов являются ни чем иным, как *аргументами* этих функций.

Работа с параметрами обеспечивается двумя элементами XSLT:
xsl:param - служит для объявления параметра;
xsl:with-param - указывает значение параметра ри вызове шаблона.

xsl:param

Синтаксис этого элемента следующий:

```
<xsl:param name="имя" select="выражение">
   <!-- Содержимое: шаблон -->
</xsl:param>
```

Элемент **xsl:template**, задающий в преобразовании шаблонное правило, может включать несколько элементов **xsl:param**, которые и будут определять параметры этого шаблона. Кроме того, **xsl:param** может быть элементов верхнего уровня - в этом случае объявляемый параметр будет *глобальным*.

Атрибут **name** задает имя параметра. Имя параметра может имет расширенную форму, например "user:param", однако, чтобы не возиться с пространствами имен, на практике всегда используют простые имена: "i", "count" или что-то в этом роде.

Параметру может быть присвоенно эначение по умолчанию.Это значение будет использованно с том случае, если параметра с таким именем шаблону при вызове переданно не было. Значение по умолчанию вычисляется следующим образом:
если в элементе xsl:param определен атрибут select, то значением по умолчанию будет результат вычисления выражения, указанного в этом атрибуте. Пример:
<xsl:param name="x" select="2 * 2" />
если атрибут select не определен, носам элемент xsl:param содержит дочерние узлы, то значением параметра по умолчанию будет фрагмент дерева, полученный в результате выполнения содержимого xsl:param. Пример:

```
<!--
   в результате будет получен фрагмент дерева
   состоящий из корневого узла и текстового узла "4"
```

```
-->
<xsl:param name="x">
   <xsl:value-of select="2 * 2" />
</xsl:param>
```

если атрибут select не определен и сам элемент xsl:param не содержит дочерних элементов, то значением по умолчанию будет пустая строка. Пример:

```
<xsl:param name="x" /> <!-- получим параметр x = " -->
```

Область видимости параметра определяется так же как область видимости переменной. Единственное, что нужно запомнить - это то, что элементы **xsl:param**, определяемые в шаблонах, должны всегда быть его **первыми дочерними элементами**. Абсолютно логично и то, что в одном шаблоне **не может быть объявленно 2 или более одноименных параметра**, так как их области видимости обязательно будут пересекаться.

xsl:with-param

Синтаксис:

```
<xsl:with-param name="имя" select="выражение">
   <!-- Содержимое: шаблон -->
</xsl:with-param>
```

Синтаксис **xsl:with-param** абсолютно идентичен синтаксису **xsl:param** и отличаются они только именем. Практически настолько же похоже и их действие, только элемент **xsl:with-param** связывает параметр со значением, которое при вызове шаблона будет использоваться **вместо** значения параметра по умолчанию. Таким образом, значение параметра, заданного в шаблоне, выбирается следующим образом:

- если в элементе, который вызывает этот шаблон, присутствует элемент xsl:with-param, передающий значение этого параметра, то в шаблоне будет использоваться переданное значение;
- в противном случае будет использоваться значение по умолчанию.

Продемонстрируем теперь использование более интересных функций - start-with(string,startSubstring) и contains(string,anySubstring). Функция start-with(string,startSubstring) проверяет, начинается ли строка string с подстроки startSubstring. Пример - ex04-4.xml, ex04-4.xsl.

Синтаксис элемента xsl:if.

<xsl:if test="starts-with($var CarMarka,$varStartWith)">

В этом элементе мы использовали переменные. Значения переменных были инициализированы ранее

```
<xsl:variable name="varStartWith"><xsl:value-of select="//letter"/></xsl:variable>
<xsl:for-each select="tutorial/avto/cars/car">
<xsl:variable name="varCarMarka"><xsl:value-of
select="carMarka"/></xsl:variable>
```

Переменная varStartWith представляет собой подстроку, с которой должны начинаться требуемые нам марки. Она не меняется, поэтому инициализируется перед циклом. Переменная varCarMarka содержит марку автомобиля, она меняется на каждом шаге цикла и, соответственно, инициализируется в теле цикла. Элемент letter XML-файла содержит букву "М".

Результат:

Марка	Номер	Цвет
Мерседес	941111	Черный
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
Мерседес	541111	Зеленый
Мазда	8946900	Белая

Функция **contains(string,anySubstring)** проверяет, содержит ли строка string подстроку **anySubstring.** Пример - ex04-5.xml, ex04-5.xsl.

Синтаксис элемента xsl:if. <xsl:if test="contains($varCarMarka,$varStartWith)">

Этот пример полностью аналогичен предыдущему. Элемент letter XML-файла содержит букву "Ш".

Результат.

Марка	Номер	Цвет
Шкода	456678	Красный
Шкода	156678	Черная

Два элемента xsl:if, вложенные друг в друга, дают нам эффект оператора AND (ex04-6.xml, ex04-6.xsl).

Соответствующий фрагмент XSL-файла.

```
<xsl:for-each select="tutorial/avto/cars/car">
  <xsl:sort order="ascending" select="number(carNumber)" data-type="number"/>
  <xsl:sort order="ascending" select="carMarka"/>
  <xsl:if test="carNumber&gt;1000000">
  <xsl:if test="carNumber&lt;2000000">
 <tr bgcolor="#11dd33">
    <td><xsl:value-of select="carMarka"/></td>
    <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
    <td><xsl:value-of select="carColor"/></td>
```

```
  </tr>
  </xsl:if>
  </xsl:if>

  </xsl:for-each>
```

Результат.

Марка	Номер	Цвет
Опель	1569875	Белый

Можно добиться и эффекта оператора OR. Для этого нам нужно включить два цикла, в каждом из которых формируется своя выборка (ex04-7.xml, ex04-7.xsl).

Соответствующий фрагмент XSL-файла.

```
<xsl:for-each select="tutorial/avto/cars/car">
  <xsl:sort order="ascending" select="number(carNumber)" data-type="number"/>
  <xsl:if test="carNumber&lt;1000000">
  <tr bgcolor="#F5F5F5">
    <td><xsl:value-of select="carMarka"/></td>
    <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
    <td><xsl:value-of select="carColor"/></td>
  </tr>
  </xsl:if>
  </xsl:for-each>
  <xsl:for-each select="tutorial/avto/cars/car">
  <xsl:sort order="ascending" select="number(carNumber)"/>
  <xsl:if test="carNumber&gt;5000000">
  <tr bgcolor="#F5F5F5">
    <td><xsl:value-of select="carMarka"/></td>
    <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
    <td><xsl:value-of select="carColor"/></td>
  </tr>
  </xsl:if>

  </xsl:for-each>
```

Если сортировка не требуется, то можно вставить два элемента xsl:if в один элемент xsl:for-each.

Элемент XSL:IF - улучшение внешнего вида таблиц

Элемент xsl:if можно применять не только для фильтрации строк выборки. Очевидно, что он может быть полезен и во многих других областях. В этом параграфе мы разберем пример использования элемента xsl:if для улучшения

внешнего вида таблицы. Заодно мы продемонстрируем реальное использование функции position(). Мы будем использовать эту функцию для того, чтобы чередовать цвет четных и нечетных строк таблицы (ex04-8.xml, ex04-8.xsl).

Фрагмент XSL-файла, который отвечает за требуемое чередование.

```
<xsl:if test="position() mod 2 = 0">
   <xsl:attribute name="bgcolor">#004EF0</xsl:attribute>
</xsl:if>
```

С элементом xsl:if и с функцией position() мы уже знакомы. Операция mod 2 дает нам остаток от деления на 2. А элемент xsl:attribute позволяет нам динамически подставлять в файл результатов различные атрибуты. Это очень мощный элемент, мы разберем еще одно применение этого элемента в следующем параграфе. А сейчас приведем для полноты картины таблицу результатов.

Марка	Номер	Цвет
БМВ	246900	Белый
БМВ	546900	Индиго
БМВ	746900	Морковный
Мазда	8946900	Белая
Мерседес	541111	Зеленый
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
Мерседес	941111	Черный
Опель	1569875	Белый
Опель	2569875	Красный
Опель	7569875	Фиолетовый
Шкода	156678	Черная
Шкода	456678	Красный

Марка	Номер	Цвет
Шкода	156678	Черная
БМВ	246900	Белый
Шкода	456678	Красный
Мерседес	541111	Зеленый
БМВ	546900	Индиго
Мерседес	641111	Оливковый
Мерседес	741111	Перломутровый
БМВ	746900	Морковный
Мерседес	941111	Черный
Опель	7569875	Фиолетовый
Мазда	8946900	Белая

К сожалению, элемент **xsl:if** в XSLT не может реализовать конструкцию "если-то-иначе". Условный оператор такого вида реализуется при помощи элементов **xsl:choose**, **xsl:when** и **xsl:otherwise**.

Примером применения xsl:choose можно так-же привести "перекрашивание" фона элемента в разные цвета в зависимости от автора т года издания. Элемент xsl:choose содержит один или несколько элементов xsl:when и необязательный элемент xsl:otherwise. При обработке xsl:choose процессор поочередно вычисляет выражения, содержащиеся в атрибутах test элементов xsl:when, приводит их к булевому типу и выполняет содержимое первого (и только первого) элемента xsl:when, тестовое выражение которого равно true. В случае если ни одно из тестовых выражений не дало "истину" в результате и в элементе xsl:choose присутствует xsl:otherwise, процессор выполнит его содержимое.

Как упоминалось ранее, при помощи элемента xsl:choose можно реализовать конструкцию вида "если-то-иначе". Делается это следующим образом:

```
<?xml version="1.0" ?>

<?xml-stylesheet type="text/xsl" href="xslchoose.xsl"?>
```

```
<catalog title="Библиотека фантастики">
        <book title="Неукротимая Планета"  author="Г.Гаррисон"   year="1985" />
        <book title="Крыса из нержавеющей стали"  author="Г.Гаррисон"
        year="1981" />
        <book title="Конные Варвары"         author="Г.Гаррисон"    year="1990" />
        <book title="Ведьмак"    author="А.Сапковски"  year="2000" />
        <book title="Властелин Колец"         author="Р.Толкиен"      year="1940" />
        <book title="Хоббит или туда и обратно"   author="Р.Толкиен"     year="1937"
/>
        <book title="Бездна голодных глаз"   author="Г.Л.Олди"      year="1997" />
        <book title="Герой должен быть один"  author="Г.Л.Олди"  year="2001" />
        <book title="Корпорация 'Бессмертие'"  author="Р.Шекли"   year="1980" />
</catalog>

<?xml version="1.0" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
   <body>
      <h3><xsl:value-of select="catalog/@title" /></h3>
      <table width="600" border="0">
         <tr style="color: white; background: black;">
            <th>Title</th>
            <th>Author</th>
            <th>Year</th>
         </tr>
         <xsl:apply-templates />
      </table>
   </body>
</xsl:template>

<xsl:template match="book">
   <tr>
      <xsl:choose>
         <xsl:when test="@year &lt; 1950">
            <xsl:attribute name="bgcolor">#FFDDDD</xsl:attribute>
         </xsl:when>
         <xsl:when test="@year &gt;= 1950 and @year &lt; 2000">
            <xsl:attribute name="bgcolor">#DDFFDD</xsl:attribute>
         </xsl:when>
         <xsl:when test="@year &gt;= 2000">
            <xsl:attribute name="bgcolor">#DDDDFF</xsl:attribute>
         </xsl:when>
      </xsl:choose>

        <td><xsl:value-of select="@title" /></td>
```

```
        <td><xsl:value-of select="@author" /></td>
        <td><xsl:value-of select="@year" /></td>
    </tr>
</xsl:template>


</xsl:stylesheet>
```

Библиотека фантастики

Title	Author	Year
Неукротимая Планета	Г.Гаррисон	1985
Крыса из нержавеющей стали	Г.Гаррисон	1981
Конные Варвары	Г.Гаррисон	1990
Ведьмак	А.Сапковски	2000
Властелин Колец	Р.Толкиен	1940
Хоббит или туда и обратно	Р.Толкиен	1937
Бездна голодных глаз	Г.Л.Олди	1997
Герой должен быть один	Г.Л.Олди	2001
Корпорация 'Бессмертие'	Р.Шекли	1980

.Динамическое формирование атрибутов на примере параметров ссылки в теге <a>

Предположим теперь, что в каждой строке таблицы нам нужно сделать ссылку на некоторую страницу и передать на эту страницу два параметра - марку и номер автомобиля. Понятно, что для каждой строки эти параметры свои, и их нельзя прописать явно в XSL-файл. Тем не менее задача легко решается при помощи элемента xsl:attribute. Мы не будем здесь строить специальный пример, ограничимся только соответствующим фрагментом XSL-файла.

```
<td>
<a target="_blank">
<xsl:attribute name="href">DisplayDetails.html?carName=
<xsl:value-of select="carMarka"/>&amp;carWeight=<xsl:value-of
select="carWeight"/></xsl:attribute>
<xsl:attribute name="title">To view some more details about <xsl:value-of
select="carMarka"/> click to car Marka</xsl:attribute>
<xsl:value-of select="carMarka"/>
</a>
</td>
```

В этом примере в ячейке таблицы мы размещаем ссылку на страницу с подробными описаниями. Ссылка указывается в атрибуте href тега <a>. Поскольку на страницу передаются два параметра, значения которых берутся из XML-файла, этот атрибут формируется динамически. Обратите также внимание - символ & (амперсанд), разделяющий передаваемые параметры, записывается в XSL-файле в виде & Во втором атрибуте нам нужна всплывающая подсказка (атрибут title), которая появляется при наведении курсора мыши на ссылку. Текст этой подсказки тоже меняется динамически. Наконец, статический атрибут target мы разместили непосредственно в теге <a>.

ПРАКТИЧЕСКАЯ РАБОТА № 2. Пример построения гистограммы

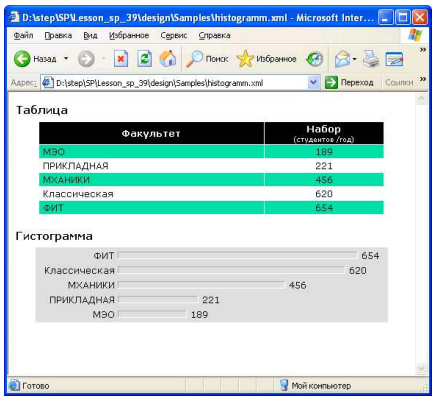


РИСУНОК 4

histogramm.xml

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="call-template1.xsl"?>
<firms>
	<firm name="ФИТ"		dohod="654"	/>
	<firm name="МЭО"		dohod="189"	/>
	<firm name="ПРИКЛАДНАЯ"		dohod="221"	/>
	<firm name="Классическая"	dohod="620"	/>
	<firm name="МХАНИКИ"		dohod="456"	/>
</firms>
```

call-template1.xsl

```
<?xml version="1.0" ?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template name="histogram">
	<table bgcolor="#dddddd" align="center">
		<colgroup style="text-align: right; padding-left: 10px;" />
		<colgroup style="padding-right: 10px;" />

		<xsl:for-each select="//firm">
			<xsl:sort select="@dohod" order="descending"/>

			<xsl:call-template name="hrow">
				<xsl:with-param name="title" select="@name" />
				<xsl:with-param name="value" select="@dohod" />
			</xsl:call-template>
		</xsl:for-each>
	</table>
</xsl:template>

<xsl:template name="hrow">
	<xsl:param name="title" />
	<xsl:param name="value" select="0" />
	<xsl:param name="coeff" select="0.5" />
```

```
        <xsl:if test="not($title = '')">
        <tr>
                <td><xsl:value-of select="$title" /></td>
                <td>
                        <span>
                                <xsl:attribute name="style">
                                        border: 1px inset;
                                        height: 10px;
                        width: <xsl:value-of select="$value * $coeff" />px;
                                        </xsl:attribute>

                        <xsl:text> </xsl:text>
                        </span>
                        <xsl:text> </xsl:text><xsl:value-of select="$value" />
                        </td>
                </tr>
        </xsl:if>
</xsl:template>

<xsl:template match="/">
   <head>
        <style>
        body, table {
                font: 10pt Verdana;
        }
        td span {
                font-size: 1pt;
        }
        h4 {
                font-size: 11pt;
                margin-bottom: 0.5em;
        }
        </style>
   </head>
   <body>

        <h4>Таблица</h4>
      <table border="0" cellspacing="1" align="center">
          <tr style="color: white; background: black;">
             <th>Факультет</th>
             <th>Набор <br /> <span style="font-weight: normal; font-size:
7pt;">(студентов /год)</span></th>
          </tr>
            <colgroup style="width: 300; padding-left: 5;" />
            <colgroup style="width: 160; text-align: center;" />
          <xsl:apply-templates select="firms/firm">
                <xsl:sort select="@dohod" />
```

```
            </xsl:apply-templates>
        </table>
          <h4>Гистограмма</h4>
          <xsl:call-template name="histogram" />
    </body>
</xsl:template>
<xsl:template match="firm">
    <tr>
          <xsl:if test="not(position() mod 2 = 0)">
          <xsl:attribute name="bgcolor"><xsl:text>#00DDA5</xsl:text></xsl:attribute>
          </xsl:if>
          <td><xsl:value-of select="@name" /></td>
          <td><xsl:value-of select="@dohod" /></td>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

На этом мы завершим рассмотрение возможностей чистого XSLT и перейдем к последнему параграфу в этом документе — к динамическому изменению содержимого Web-страницы при помощи возможностей JavaScript и XML/XSLT без каких-либо дополнительных обращений к базе данных.

ПРАКТИЧЕСКАЯ РАБОТА № 3. Стилевые таблицы XSL, JavaScript и XML

Объединим теперь наши знания XML с возможностями, которые нам предоставляет JavaScript. Предположим, что нам нужно иметь возможность динамически изменять сортировку столбцов таблицы при щелчке на заголовке того или иного столбца. Понятно, что для этого нам нужно иметь один XML-файл, содержащий строки таблицы, несколько XSL-файлов, каждый из которых содержит требуемую сортировку и нечто, что объединит это все вместе и заставит работать.

Перейдем к реализации этой программы.

В качестве XML-файла возьмем привычный нам файл со списком машин - ex05-1.xml. Обратите внимание - мы убрали из файла ссылку на XSL-файл - нам нужно менять шаблон преобразования динамически.

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<tutorial>
<avto>
 <cars>
  <carsCaption>Автомобили</carsCaption>
  <carsCaptionMarka>Марка</carsCaptionMarka>
  <carsCaptionNumber>Номер</carsCaptionNumber>
      <carsCaptionColor>Цвет</carsCaptionColor>
       <car>
        <carMarka>Мерседес</carMarka>
        <carNumber>341111</carNumber>
        <carColor>Черный</carColor>
       </car>
       <car>
        <carMarka>Шкода</carMarka>
        <carNumber>456678</carNumber>
        <carColor>Красный</carColor>
       </car>
       <car>
        <carMarka>Опель</carMarka>
        <carNumber>1569875</carNumber>
        <carColor>Белый</carColor>
       </car>
       <car>
        <carMarka>БМВ</carMarka>
        <carNumber>546900</carNumber>
        <carColor>Индиго</carColor>
       </car>

  </cars>
</avto>

</tutorial>
```

Создадим также три XSL-файла, в каждом из которых у нас будет свой элемент xsl:sort, задающий сортировку строк - ex05-1a.xsl, ex05-1b.xsl, ex05-1c.xsl.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<table border="1">
  <tr bgcolor="#CCCCCC">
    <td align="center"><strong><a href="#" onClick="orderByMarka();">
                        <xsl:value-of select="//carsCaptionMarka"/>
                        </a></strong>
    </td>
    <td align="center"><strong><a href="#" onClick="orderByNumber();">
                        <xsl:value-of select="//carsCaptionNumber"/>
                        </a></strong>
    </td>
    <td align="center"><strong><a href="#" onClick="orderByColor();">
                        <xsl:value-of select="//carsCaptionColor"/>
                        </a></strong>
    </td>
 </tr>
  <xsl:for-each select="tutorial/avto/cars/car">
  <xsl:sort order="ascending" select="carMarka"/>
  <tr bgcolor="#F5F5F5">
    <td><xsl:value-of select="carMarka"/></td>
    <td align="right"><xsl:value-of select="carNumber"/> <xsl:value-of
select="carNumber/@caption"/></td>
    <td><xsl:value-of select="carColor"/></td>
  </tr>
  </xsl:for-each>
</table>
</xsl:template>

</xsl:stylesheet>
```

Приведем здесь текст элемента xsl:sort для каждого файла

```
<xsl:sort order="ascending" select="carMarka"/>
<xsl:sort order="ascending" select="number(carNumber)" data-
type="number"/>
<xsl:sort order="ascending" select="carColor"/>
```

Теперь нам осталось только объединить все это вместе. Ниже мы полностью приводим текст файла ex05-1.htm, сопроводив его необходимыми комментариями.

```
<html>
<head>
<script language="JavaScript">
var source;
var style;
```

Функция инициализации необходимых объектов. В этой же функции выводится первоначальный вариант на экран.

```
function init() {
```

Создаем объект для файла - источника данных.

```
source = new ActiveXObject("Microsoft.XMLDOM");
source.async = false;
```
Создаем объект для файла с шаблоном преобразования (для файла стиля).
```
style = new ActiveXObject("Microsoft.XMLDOM");
style.async = false;
```
Загружаем записи в файл - источник данных.Записи берем из существующего XML-файла.
```
source.load("ex05-1.xml");
```
Загружаем файл стиля. Первоначальная сортировка - по цвету.
```
style.load("ex05-1a.xsl");
```
Теперь нам нужно вывести информацию на экран. Внимательно проанализируйте синтаксис и запомните его.
```
document.all.item("xslresult").innerHTML = source.transformNode(style);
return true;
}
```
Сортируем записи по маркее.
```
function orderByMarka() {
style.load("ex05-1a.xsl");
document.all.item("xslresult").innerHTML = source.transformNode(style);
return true;
}
```
Сортируем записи по номеру.
```
function orderByNumber() {
style.load("ex05-1b.xsl");
document.all.item("xslresult").innerHTML = source.transformNode(style);
return true;
}
```
Сортируем записи по цвету.
```
function orderByColor() {
style.load("ex05-1c.xsl");
document.all.item("xslresult").innerHTML = source.transformNode(style);
return true;}
</script>
</head>
```
При загрузке страницы создадим все необходимые объекты и выведем первоначальный вариант на экран.
```
<body onLoad="init()">
<div id="xslresult">
<!-- Здесь будет размещаться окончательный вариант HTML-содержимого  -->
</div>
</body>
</html>
```
Мы добились своей цели - при щелчке мышью на заголовке столбца строки сортируются в соответствии со значениями в выбранном столбце.
Подключим каскадную таблицу стилей:
```
<link  rel="stylesheet" type="text/css" href="my.css"  />
```

my.css

Марка	Номер	Цвет
БМВ	546900	Индиго
Мерседес	341111	Черный

```
a:link {color: rose;
        width: 200px;
            border: outset 4px grey;
            text-decoration: none;
            text-align: center;
            background-color: blue;}
a:visited {color:rose;
        width: 200px;
            border: outset 4px grey;
            text-decoration: none;
            text-align: center;
            }
a:active {color: rose;
        width:200px;
            border: inset 4px grey;
            text-decoration: none;
            text-align: center;
            background-color: blue;}
a:hover {color: pink;
        width: 200px;
            border: outset 4px grey;
            text-decoration: none;
            text-align: center;
            background-color: gray;}

body {
    font-family: Minion Web, Palatino Linotype, Book Antiqua, Baskerville Win95BT,
Times New Roman, serif;
         background-color: navy;
            background-image: url(fon.jpg);

  }

div
{
      border-right: 4px outset; border-top: 4px outset;
      padding:100px;   margin:10px;
      float: right;background-image: url(fon2.jpg);
      overflow: scroll;   text-transform: uppercase;
      border-left: 4px outset;width: 700px;
      height: 300px;color: #0099cc;
      border-bottom: 4px outset;
      font-style: italic;   position: absolute;
      top: 100px;left:100px;
      text-align: left;
}
```

Связывание данных является первым из методов отображения XML-документа с традиционной HTML-страницы, с которым вы познакомитесь. Отображение XML с HTML-страниц имеет все преимущества хранения данных в XML-документе, с его гибким синтаксисом для структурирования данных и маркировки каждого фрагмента информации, не теряя богатых возможностей форматирования и динамического программирования HTML.

При связывании данных вы соединяете XML-документ с HTML-страницей, а затем встраиваете стандартные HTML-элементы, такие как SPAN или TABLE, в отдельные XML-элементы. HTML-элементы затем автоматически отображают содержимое XML-элементов, в которые они встроены.

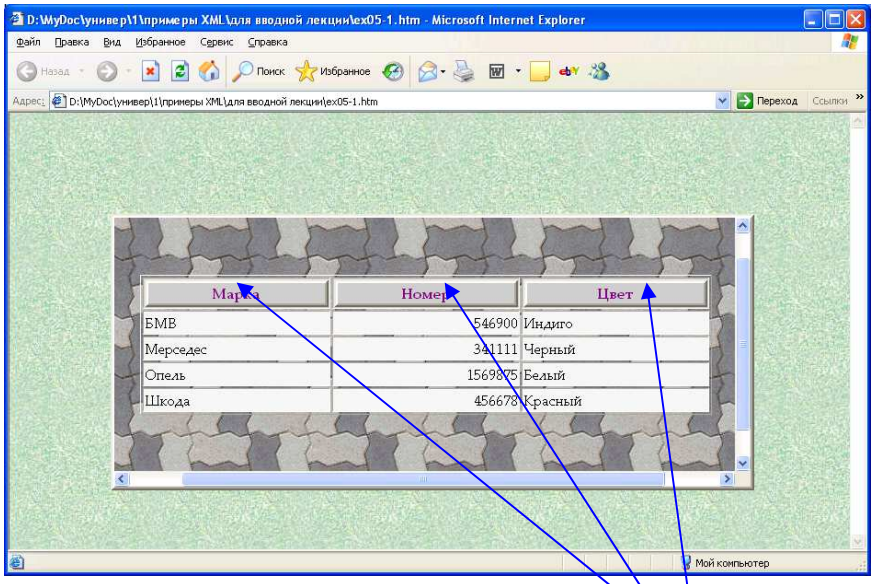


РИСУНОК 5

```
<tr bgcolor="#CCCCCC">
   <td align="center"><strong><a href="#" onClick="orderByMarka();">
                        <xsl:value-of select="//carsCaptionMarka"/>
                        </a></strong>
   </td>
   <td align="center"><strong><a href="#" onClick="orderByNumber();">
                        <xsl:value-of select="//carsCaptionNumber"/>
                        </a></strong>
   </td>
   <td align="center"><strong><a href="#" onClick="orderByColor();">
                        <xsl:value-of select="//carsCaptionColor"/>
                        </a></strong>
   </td>
 </tr>
```

РИСУНОК 6

Основные шаги при связывании данных

Установка связи XML-документа с HTML-страницей, на которой вы хотите отобразить данные XML. Этот шаг обычно реализуется включением HTML-элемента с именем XML в HTML-страницу. Например, следующий элемент на HTML-странице связывает XML-документ Book.xml со страницей:

<XML ID="dsoBook" SRC="Book.xml"X/XML>

Сцепление HTML элементов с XML-элементами. Когда вы сцепляете HTML-элементы с XML-элементом, HTML-элемент автоматически отображает содержимое XML-элемента. Например, следующий элемент SPAN на HTML-странице сцеплен с элементом AUTHOR связанного XML-документа:

<SPAN DATASRC="#dSoBook" DATAFLD="AUTHOR"X/SPAN>

В результате HTML-элемент SPAN отображает содержимое XML-элемента AUTHOR.

Базовая технология связывания данных в действительности столь же проста, как в этом примере, хотя, в дальнейшем вы познакомитесь с различными вариациями и способами использования этой технологии.

Установка связи XML-документа с HTML-страницей

Чтобы отобразить XML-документ на HTML-странице, вы должны установить его связь со страницей. Самый простой путь сделать это в Microsoft Internet Explorer - включить в страницу HTML-элемент с именем XML, так называемый фрагмент данных. Вы можете использовать одну из двух различных форм записи для фрагмента данных.

В первой форме весь текст XML-документа помещается между начальным и конечным тегами XML. Вот пример фрагмента данных на следующей HTML-странице:

```
<HTML>
<HEAD>
<TITLE>Book Description<TITLE>
</HEAD>
<BODY>
<XML ID="dsoBook">
<?xml version="1.0"?>
<BOOK>
<TITLE>The Adventures of Huckleberry Нпп<TITLE>
<AUTHOR>Mark Twain</AUTHOR>
<BINDING>mass market paperback</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
</BOOK>  </XML>
<!-- другие элементы HTML... -->
</BODY>  </HTML>
```

Во второй форме записи HTML-элемент с именем XML остается пустым и содержит только URL XML-документа. Вот пример фрагмента данных на HTML-странице:

```
<HTML>
<HEAD>
<TITLE>Book Description</TITLE>
</HEAD>
<BODY>
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

```
<!-- другие элементы HTML... -->
</BODY>
</HTML>
```

В предыдущем примере текст XML-документа должен содержаться в отдельном файле Book.xml:

```
<?xml version="1.0"?>
<!-- Имя файла: Book.xml -->
<BOOK>
<TITLE>The Adventures of Huckleberry Рпп</TITLE>
<AUTHOR>Mark Twain</AUTHOR>
<6INDING>mass market paperback</BINDING>
<PAGES>298</PAGES>
<PRICE>$5.49</PRICE>
</BOOK>
```

Вторая форма более соответствует основам философии XML, согласно которой собственно данные (XML-документ) хранятся отдельно от информации по их форматированию и обработке (таблицы стилей или, в данном случае, HTML-страницы). Вторая форма облегчает работу с XML-документом особенно если один документ отображается на нескольких различных HTML-страницах. В рассматриваемых в этой книге примерах вы будете иметь дело только со второй формой.

Вы должны присвоить атрибуту ID фрагмента данных уникальный идентификатор, который используется для доступа к XML-документу с I ITML-страницы При второй форме записи фрагмента данных вы присваиваете атрибуту SRC URL файла, содержащего данные XML. Вы можете использовать URL, как и следующем примере:

```
<XML ID="dsoBook" SRC=http://WWW.my_domain.com/documents/Book.xml>
</XML>
```

Однако чаще используется частичный URL, который задает местонахождение относительно местонахождения HTML-страницы, содержащей фрагмент данных. Например, атрибут SRC в следующем фрагменте данных указывает, что файл Book.xml находится в той же папке, что и HTML-страница:

```
<XML ID="dsoBook" SRC="Book.xml"></XML>
```

Относительные URL более распространены, потому что XML-документ обычно содержится в той же папке, что и HTML-страница, либо в одной из вложенных папок.

Как хранятся данные XML

Когда Internet Explorer 5 открывает HTML-страницу, его встроенный XML-процессор синтаксически анализирует XML-документ. Internet Explorer 5 также создает программный объект, который носит название Объект исходных данных (Data Source Object DSO), который хранит данные XML и обеспечивает доступ к этим данным. DSO хранит данные XML как набор записей — т.е. множество записей и их полей. Например, если вы включите документ Inventory.xml (см. Листинг 8-1) в страницу как фрагмент данных, DSO будет хранить каждый элемент

BOOK как запись, а каждый дочерний элемент внутри BOOK (TITLE, AUTHOR и т.д.) как поле.

Когда вы сцепляете HTML-элемент с XML-элементом, DSO автоматически предоставляет значение XML-элемента и управляет всеми его свойствами. DSO также позволяет вам напрямую осуществлять доступ и манипулирование имеющимся набором записей посредством ряда методов, свойств и событий. Методы представляют собой функции, которые вы можете вызывать со страницы для доступа или модификации набора записей. (Например, вы можете использовать методы для перемещения между записями.) Свойства представляют собой установленные на данный момент параметры, которые вы можете считывать и в ряде случаев изменять со страницы. (Например, вы можете считать свойство, которое сообщает вам, достигли ли вы последней записи.) События представляют собой определенные смены состояний (например, изменение значений записи), которыми вы можете управлять посредством функции сценария, который вы создаете для страницы.

На странице идентификатор, который вы присваиваете атрибуту ID во фрагменте данных, представляет DSO. (В предыдущем разделе в рассмотренном примере таким идентификатором является dsoBook)

Проверка на наличие ошибок XML

Когда вы открываете XML-документ (автономный или с таблицей стилей) непосредственно в Internet Explorer 5, браузер проверяет, является ли документ корректно сформированным. Если он обнаруживает ошибки, то приостанавливает отображение документа и выводит сообщение о фатальной ошибке, которое помогает вам выявить ошибку и устранить ее.

Если вы открываете XML-документ через фрагмент данных на HTML-странице, Internet Explorer 5 проверяет, является ли документ корректно сформированным, а также является ли он валидным (если документ включает объявление типа документа). Но если документ содержит ошибку, Internet Explorer просто не будет отображать данные XML, не выводя сообщение об ошибке.

Сцепление HTML-элементов с XML-элементами

Вы можете осуществлять сцепление HTML-элементов с XML-элементами двумя основными способами.

Табличное сцепление, что означает сцепление HTML-элемента TABLE с данными XML, так что в таблице автоматически отображается весь набор записей, принадлежащих XML-документу.

Сцепление по отдельным записям, что означает сцепление не табличных элементов HTML (например, элементов SPAN) с XML-элементами таким образом, что за один раз отображается только одна запись.

Использование табличного сцепления данных

Самый простой способ отобразить XML-документ, который состоит из группы записей (такой как lnventory.xml, представленный в Листинге 8-1) — это сцепить HTML-элемент TABLE с данными XML таким образом, чтобы в таблице автоматически отображались сразу все записи (или одна страница записей за раз, если вы установили режим постраничного отображения). При таком подходе Internet Explorer 5 берет на себя большую часть работы, вам не нужно писать сценарии или вызывать методы (функции). (Одно исключение состоит в том, что если вы выбрали

режим пролистывания, вам потребуется включить несколько вызовов простых функций)

Вы можете использовать одну таблицу HTML для отображения XML-документа, структурированного как набор записей, либо вы можете использовать вложенные HTML-таблицы для отображения XML-документа, содержащего иерархический набор записей (более сложную структуру записей).

Использование одной HTML-таблицы для отображения простого набора записей

Вы можете использовать один HTML-элемент TABLE для отображения XML-документа, в котором данные организованы в виде простого набора записей - т.е. XML-документа, составленного следующим образом:

корневой элемент содержит множество элементов типа запись (record) (подобные элементы иногда называются просто записями);

каждый элемент типа запись содержит одинаковый набор элементов типа поле (field) (в этой главе подобные элементы иногда называются просто полями)', каждый элемент типа поле содержит только символьные данные. Если дочерний элемент элемента запись содержит один или несколько своих собственных дочерних элементов, DSO интерпретирует его как вложенную запись, а не как поле.

Примером такого типа XML-документов является документ lnventory.xml. В этом документе корневой элемент (INVENTORY) содержит набор из восьми элементов-записей (элементы BOOK), и каждый из элементов-записей имеет одинаковый набор элементов-полей, которые содержат только символьные данные (TITLE, AUTHOR, BINDING, PAGES, PRICE).

Inventory.xml

```
<?xml version="1.0"?>
<!-- File Name: Inventory.xml -->
<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Leaves of Grass</TITLE>
    <AUTHOR>Walt Whitman</AUTHOR>
    <BINDING>hardcover</BINDING>
    <PAGES>462</PAGES>
    <PRICE>$7.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Legend of Sleepy Hollow</TITLE>
    <AUTHOR>Washington Irving</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>98</PAGES>
    <PRICE>$2.95</PRICE>
```

```
  </BOOK>
  <BOOK>
    <TITLE>The Marble Faun</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>473</PAGES>
    <PRICE>$10.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Moby-Dick</TITLE>
    <AUTHOR>Herman Melville</AUTHOR>

    <BINDING>hardcover</BINDING>
    <PAGES>724</PAGES>
    <PRICE>$9.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Portrait of a Lady</TITLE>
    <AUTHOR>Henry James</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>256</PAGES>
    <PRICE>$4.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Scarlet Letter</TITLE>
    <AUTHOR>Nathaniel Hawthorne</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>253</PAGES>
    <PRICE>$4.25</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Turn of the Screw</TITLE>
    <AUTHOR>Henry James</AUTHOR>
    <BINDING>trade paperback</BINDING>
    <PAGES>384</PAGES>
    <PRICE>$3.35</PRICE>
  </BOOK>
    </INVENTORY>
```

Когда вы связываете таблицу с XML-документом, данные, принадлежащие каждому из элементов записей, отображаются в отдельной строке таблицы, а каждый из дочерних элементов полей — в отдельном столбце.

В качестве примера возьмем HTML-страницу **Inventory Table.htm**, которая содержит таблицу, сцепленную с данными документа Inventory.xml.

Inventory Table.htm

```
<!-- File Name: Inventory Table.htm -->
    <HTML><HEAD>  <TITLE>Book Inventory</TITLE></HEAD><BODY>
      <XML ID="dsoInventory" SRC="Inventory.xml"></XML>
```

```
<H2>Book Inventory</H2>

<TABLE DATASRC="#dsoInventory" BORDER="1" CELLPADDING="5">
  <THEAD>
    <TH>Title</TH>
    <TH>Author</TH>
    <TH>Binding</TH>
    <TH>Pages</TH>
    <TH>Price</TH>
  </THEAD>
  <TR ALIGN="center">
    <TD><SPAN DATAFLD="TITLE"
      STYLE="font-style:italic"></SPAN></TD>
    <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
    <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
    <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
    <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

XML-документ связан с HTML-страницей посредством фрагмента данных на этой странице, имеющего ID dsolnventory:

```
<XML ID="dsoInventory" SRC="lnventory.xml"></XML>
```

Элемент TABLE страницы сцеплен со всем XML-документом путем присвоения атрибуту DATASRC элемента идентификатора (ID) фрагмента данных, предваренного символом #:

```
<TABLE DATASRC="#dsolnventory" BORDER="1" CELLPADDING="5">
```

Таблица определена со стандартным заголовком (элемент THEAD) и с одной строкой (элемент TR). Каждая ячейка в этой строке (т.е., каждый элемент TD) состоит из элемента SPAN, который сцеплен с одним из полей XML-документа таким образом, что этот элемент отображает содержимое поля. Например, первая ячейка содержит элемент SPAN, сцепленный с полем TITLE:

```
<TD><SPAN DATAFLD="TITLE"
STYLE="font-style:italic"></SPAN></TD>
```

Элемент SPAN связывается с полем XML путем присвоения имени поля (в данном примере TITLE) атрибуту DATAFLD элемента.

Вот как работает связывание данных: даже если в элементе TABLE определена только одна строка, когда браузер отображает таблицу, он повторяет строковый элемент для каждой записи в XML-документе. В первой строке, следующей за заголовком, отображены поля (TITLE, AUTHOR и т.д.), принадлежащие первой записи (элемент BOOK для книги Adventures of Huckleberry Finn). В следующей строке отображены поля для второй записи (элемент BOOK для книги Leaves of Grass) и т.д.

У вас может возникнуть вопрос, почему ячейки (элементы TD) не сцеплены непосредственно с полями XML. Ответ заключается в том, что элемент TD не

является связываемым HTML-элементом. Следовательно, вы должны включить внутрь каждого элемента TD связываемый элемент (обычно SPAN).

Ссылка. Чтобы получить информацию о том, как работает HTML и динамический HTML (DHTML) применительно к Internet Explorer, посетите следующий Web-сайт MSDN: http://msdn.microsoft.com/workshop/author/default.asp. Чтобы познакомиться с официальной спецификацией HTML, обратитесь к Web-сайту консорциума W3C по адресу http://www.w3.org/TR/REC-html40/.

Использование постраничного отображения

Если XML-документ содержит много записей, вы можете использовать постраничный вывод группы записей за один раз вместо отображения всех записей одновременно в огромной таблице. Для активизации постраничного отображения в обычной связанной таблице, выполните следующие действия.

Установите для атрибута DATAPAGESIZE сцепленного элемента TABLE значение, равное максимальному числу записей, которые вы хотите отобразить за раз. Каждая страница записей будет содержать заданное вами число записей. Например, следующий начальный тег для элемента TABLE присваивает число «5» атрибуту DATAPAGESIZE, в результате чего в таблице будет отображено пять записей за раз:

<TABLE DATASRC="#dsolnventory" DATAPAGESIZE="5">

Присвойте уникальный идентификатор атрибуту ID элемента TABLE, как для следующего начального тега:

<TABLE ID="lnventoryTabte" DATASRC="#dsolnventory" DATAPAGESIZE="5">

Чтобы перемещаться между записями, вызывайте методы элемента TABLE, приведенные в таблице. Для приведенных в последнем столбце примеров предполагается, что таблица имеет идентификатор InventoryTable.

Таблица 8.1

Метод элемента TABLE	Эффект	Пример вызова
firstPage	Отображает первую страницу записей	InventoryTable.firstPage()
previousPage	Отображает предыдущую страницу записей	InventoryTable.previousPage()
nextPage	Отображает следующую страницу записей	InventoryTable.nextPage()
lastPaoe.	Отображает последнюю страницу записей	InvontoryTable.lastPagc(),

Если в текущий момент отображена первая страница, вызов метода previousPage игнорируется, а если отображена последняя страница, то игнорируется вызов neoctPage.

Вы можете вызвать любой из этих методов из написанного вами сценария. Однако наиболее простой способ обращения к одному из методов заключается в присвоении метода атрибуту ONCLICK HTML-элемента BUTTON, как в следующем примере:

<BUTTON ONCLICK="lnventoryTable.nextPage()">Next Page</BUTTON>

Этот элемент отображает кнопку. Когда пользователь щелкает на кнопке, вызывается метод, присвоенный атрибуту ONCLICK, а именно, InventoryTable.nextPage.

В верхней части страницы имеется четыре элемента BUTTON, каждый из которых выполняет действие в соответствии с методами постраничного вывода таблицы. Когда вы впервые открываете HTML-страницу, в таблице отображаются первые пять записей. Щелчок мышью на кнопке Next Page приводит к отображению следующих пяти записей (или, в конце таблицы, оставшегося числа записей), а щелчок на кнопке Previous приводит к отображению предыдущих пяти записей (или, в начале таблицы, первых пяти записей). Щелчок на кнопке First Page или на кнопке Last Page приводит к отображению первых или последних пяти записей.

Inventory Big.xml

```
<?xml version="1.0"?>

<!-- File Name: Inventory Big.xml -->

<INVENTORY>
  <BOOK>
    <TITLE>The Adventures of Huckleberry Finn</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>298</PAGES>
    <PRICE>$5.49</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Adventures of Tom Sawyer</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>205</PAGES>
    <PRICE>$4.75</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>The Ambassadors</TITLE>
    <AUTHOR>Henry James</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>305</PAGES>
    <PRICE>$5.95</PRICE>
  </BOOK>
```

```
<BOOK>
  <TITLE>The Awakening</TITLE>
  <AUTHOR>Kate Chopin</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>195</PAGES>
  <PRICE>$4.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Billy Budd</TITLE>
  <AUTHOR>Herman Melville</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>195</PAGES>
  <PRICE>$4.49</PRICE>
</BOOK>
<BOOK>
  <TITLE>A Connecticut Yankee in King Arthur's Court</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>385</PAGES>
  <PRICE>$5.49</PRICE>
</BOOK>
<BOOK>
  <TITLE>Joan of Arc</TITLE>
  <AUTHOR>Mark Twain</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>465</PAGES>
  <PRICE>$6.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>Leaves of Grass</TITLE>
  <AUTHOR>Walt Whitman</AUTHOR>
  <BINDING>hardcover</BINDING>
  <PAGES>462</PAGES>
  <PRICE>$7.75</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Legend of Sleepy Hollow</TITLE>
  <AUTHOR>Washington Irving</AUTHOR>
  <BINDING>mass market paperback</BINDING>
  <PAGES>98</PAGES>
  <PRICE>$2.95</PRICE>
</BOOK>
<BOOK>
  <TITLE>The Marble Faun</TITLE>
  <AUTHOR>Nathaniel Hawthorne</AUTHOR>
  <BINDING>trade paperback</BINDING>
  <PAGES>473</PAGES>
  <PRICE>$10.95</PRICE>
```

```
  </BOOK>
  <BOOK>
    <TITLE>The Portrait of a Lady</TITLE>
    <AUTHOR>Henry James</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>256</PAGES>
    <PRICE>$4.95</PRICE>
  </BOOK>
  <BOOK>
    <TITLE>Roughing It</TITLE>
    <AUTHOR>Mark Twain</AUTHOR>
    <BINDING>mass market paperback</BINDING>
    <PAGES>324</PAGES>
    <PRICE>$5.25</PRICE>
  </BOOK>
</INVENTORY>
```

Inventory Big Table.htm

```
<!-- File Name: Inventory Big Table.htm -->

<HTML>
<HEAD>
  <TITLE>Book Inventory</TITLE>
</HEAD>

<BODY>
  <XML ID="dsoInventory" SRC="Inventory Big.xml"></XML>
  <H2>Book Inventory</H2>
  <BUTTON ONCLICK="InventoryTable.firstPage()">
    |&lt; First Page
  </BUTTON>
    
  <BUTTON ONCLICK="InventoryTable.previousPage()">
    &lt; Previous Page
  </BUTTON>
    
  <BUTTON ONCLICK="InventoryTable.nextPage()">
    Next Page &gt;
  </BUTTON>
    
  <BUTTON ONCLICK="InventoryTable.lastPage()">
    Last Page &gt;|
  </BUTTON>
  <p>

  <TABLE ID="InventoryTable" DATASRC="#dsoInventory"
```

```
    DATAPAGESIZE="5" BORDER="1" CELLPADDING="5">
    <THEAD>
      <TH>Title</TH>
      <TH>Author</TH>
      <TH>Binding</TH>
      <TH>Pages</TH>
      <TH>Price</TH>
    </THEAD>
    <TR ALIGN="center">
      <TD><SPAN DATAFLD="TITLE"
        STYLE="font-style:italic"></SPAN></TD>
      <TD><SPAN DATAFLD="AUTHOR"></SPAN></TD>
      <TD><SPAN DATAFLD="BINDING"></SPAN></TD>
      <TD><SPAN DATAFLD="PAGES"></SPAN></TD>
      <TD><SPAN DATAFLD="PRICE"></SPAN></TD>
    </TR>
  </TABLE>
</BODY>
</HTML>
```

Во второй части лабораторного практикума мы познакомимся с основными принципами XML DOM и методами программной обработки XML документов путем манипулирования узлами дерева документа, а также с технологией AJAX. Рассмотрим возможности использования XML файлов для организации хранилищ данных в DotNet.

ЛИТЕРАТУРА

1. Бумфрей Ф., Дирецо О., Дакетт Й. и др
   XML. Новые преспективы WWW М.:ДМК, 2000, -688 с
2. Марк Зайден
   XML для электронной коммерции Издательство: Бином. Лаборатория знаний, 2003, 480 с
3. **Язык XML - практическое введение**
   http://www.citforum.ru/internet/xml/index.shtml
4. Мартин Д., Бирбек М., Кэй М. и др.
   XML для профессионалов. - М.: Лори, 2001. - 900 с.
5. О.Н. Кищенко. **Языки информационного обмена**
   http://www.intuit.ru/department/internet/lande/

Учебное издание

Геренко Ольга Андреевна,
Розновец Ольга Игоревна,
Пенко Елена Анатольевна

Методическое пособие по курсу «Язык разметки XML. Часть 1»

для студентов специальности «Компьютерные системы и сети»

Издано в авторской редакции

Підп. до друку 20.05.2010. Формат 60х84/16.
Гарн. Таймс. Тираж 50 прим.

Редакційно-видавничий Центр
Одеського національного університету
імені І.І. Мечникова,
65082, м. Одеса, вул. Єлісаветинська, 12, Україна
Тел.: (048) 723 28 39