Сэл Мангано

XSLT
СБОРНИК РЕЦЕПТОВ

Москва, Санкт-Петербург, 2008

УДК 004.438
ББК 32.973.26-018.2
Г26

Мангано Сэл
Г26 XSLT. Сборник рецептов. – М.: ДМК Пресс, СПБ.: БХВ-Петербург, 2008. – 864 с.: ил.

ISBN 978-5-94074-419-1 («ДМК Пресс»)
ISBN («БХВ-Петербург»)

Язык XSLT (Extensible Stylesheet Language Transformation) стал основным инструментом обработки XML-документов, но многие разработчики все еще не освоили его в полной мере и потому считают, что проще модифицировать имеющийся код, чем писать новый с нуля. В версии 2.0 многие проблемы решены, но появился ряд новых возможностей, которые еще надо изучить. К тому же она пока недостаточно поддержана.

Во втором издании настоящей книги приведены сотни решений задач, с которыми регулярно сталкиваются программисты. Даются варианты для обеих версий XSLT. Диапазон рецептов чрезвычайно широк: от операций со строками и математических вычислений до таких сложных тем, как расширение XSLT, тестирование и отладка таблиц стилей и создание графики в форме SVG. В каждом рецепте обосновывается выбор решения и объясняется примененная техника. Для многих задач приводятся альтернативные решения с замечаниями по поводу удобства пользования и производительности.

Предлагая рецепты, рассчитанные на разные уровни квалификации, эта книга станет идеальным спутником программиста, который любит учиться на примерах. Неважно, примериваетесь вы к XSLT впервые или уже знакомы с этим языком и хотите иметь подборку готовых рецептов для решения сложных задач, в ней вы найдете самые разные способы применения XSLT.

УДК 004.438
ББК 32.973.26-018.2

ISBN 978-5-94074-419-1 («ДМК Пресс»)
ISBN («БХВ-Петербург»)

Содержание

Предисловие 9
О структуре книги 11
Типографские соглашения 13
О примерах кода 14
Благодарности 15

Глава 1. Язык XPath 17
1.1. Применение осей 18
1.2. Фильтрация узлов 24
1.3. Работа с последовательностями 27
1.4. Включение условий в выражения if 29
1.5. Исключение рекурсии с помощью выражений for 32
1.6. Упрощение сложной логики с помощью кванторов 34
1.7. Операции над множествами 36
1.8. Сравнение узлов 37
1.9. Как ужиться с расширенной системой типов в XPath 2.0 38
1.10. Как воспользоваться расширенной системой типов в XPath 2.0 41

Глава 2. Строки 43
2.0. Введение 43
2.1. Завершается ли данная строка указанной подстрокой? 44
2.2. Нахождение позиции начала подстроки 44
2.3. Удаление заданных символов из строки 45
2.4. Поиск подстроки с конца строки 49
2.5. Повторение строки N раз 53
2.6. Обращение строки 57
2.7. Замена текста 61
2.8. Преобразование регистра 67
2.9. Разбиение строки на лексемы 70
2.10. Как обойтись без регулярных выражений 73
2.11. Использование регулярных выражений 74
2.12. Расширения EXSLT для работы со строками 76

Глава 3. Математические операции над числами 80
3.0. Введение 80
3.1. Форматирование чисел 81
3.2. Округление чисел с заданной точностью 90

3.3. Преобразование римских числительных в числа 93
3.4. Преобразование из одной системы счисления в другую 96
3.5. Реализация стандартных математических функций 100
3.6. Вычисление сумм и произведений 114
3.7. Нахождение минимума и максимума 120
3.8. Вычисление статистических функций 128
3.9. Вычисление комбинаторных функций 132
3.10. Проверка битов 134

Глава 4. Даты и время 141
4.0. Введение 141
4.1. Вычисление дня недели 143
4.2. Вычисление последнего дня месяца 145
4.3. Получение названий дней и месяцев 146
4.4. Вычисление юлианского и абсолютного дня, соответствующих заданной дате 151
4.5. Вычисление номера недели, соответствующего заданной дате 156
4.6. Юлианский календарь 157
4.7. Календарь ISO 158
4.8. Исламский календарь 161
4.9. Еврейский календарь 164
4.10. Форматирование даты и времени 174
4.11. Определение светских и церковных праздников 187

Глава 5. Отбор и обход 191
5.0. Введение 192
5.1. Игнорирование элементов-дубликатов 194
5.2. Отбор всех элементов, кроме одного 200
5.3. Отбор узлов по контексту 201
5.4. Выполнение обхода в прямом порядке 203
5.5. Выполнение обхода в обратном порядке 208
5.6. Выполнение обхода во внутреннем порядке 210
5.7. Выполнение обхода по уровням 214
5.8. Обработка узлов по позиции 221

Глава 6. XSLT 2.0 229
6.0. Введение 229
6.1. Преобразование простых именованных шаблонов в функции XSLT 229
6.2. Для группировки пользуйтесь командой for-each-group, а не методом Мюнха 231

6.3. Модульность и режимы 235
6.4. Применение типов в целях безопасности и точности выражения намерений 236
6.5. Избегайте подводных камней при переносе с XSLT 1.0 на 2.0 238
6.6. Эмуляция объектно-ориентированных методик повторного использования и паттернов проектирования 240
6.7. Обработка неструктурированного текста с помощью регулярных выражений 245
6.8. Решение трудных задач сериализации с помощью таблиц символов 249
6.9. Вывод нескольких документов 251
6.10. Обработка строковых литералов, содержащих кавычки 253
6.11. Новые возможности старых конструкций XSLT 1.0 253

Глава 7. Преобразование XML в текст 260
7.0. Введение 260
7.1. Пустое пространство 261
7.2. Экспорт XML в файл с разделителями полей 267
7.3. Создание отчета с несколькими колонками 287
7.4. Отображение иерархии 299
7.5. Вывод текста с нумерацией 308
7.6. Вывод текста в области заданной ширины с заданным выравниванием 319

Глава 8. Преобразование XML в XML 323
8.0. Введение 323
8.1. Преобразование атрибутов в элементы 324
8.2. Преобразование элементов в атрибуты 327
8.3. Переименование элементов или атрибутов 332
8.4. Объединение документов с одной и той же схемой 339
8.5. Объединение документов с различными схемами 345
8.6. Расщепление документов 351
8.7. Уплощение иерархии XML 353
8.8. Углубление иерархии XML 357
8.9. Реорганизация иерархии XML 363

Глава 9. Опрос XML-документа 368
9.0. Введение 369
9.1. Выполнение теоретико-множественных операций над наборами узлов 370

9.2. Выполнение теоретико-множественных операций над наборами узлов с использованием семантики значений ... 373
9.3. Сравнение наборов узлов на равенство по значению ... 385
9.4. Выполнение запросов, сохраняющих структуру ... 390
9.5. Соединения ... 393
9.6. Реализация на XSLT сценариев, приведенных в спецификации W3C XML Query ... 398

Глава 10. Преобразование XML в HTML ... 433
10.0. Введение ... 433
10.1. Использование XSLT в качестве языка стилизации ... 434
10.2. Создание документов, связанных гиперссылками ... 443
10.3. Создание HTML-таблиц ... 446
10.4. Создание фреймов ... 455
10.5. Создание таблиц стилей, управляемых данными ... 461
10.6. Создание замкнутого преобразования ... 468
10.7. Заполнение формы ... 473

Глава 11. Преобразование XML в SVG ... 480
11.0. Введение ... 480
11.1. Преобразование имеющейся заготовки SVG ... 482
11.2. Создание повторно используемых утилит генерации SVG для графиков и диаграмм ... 490
11.3. Создание графического представления деревьев ... 532
11.4. Создание интерактивных Web-страниц, включающих SVG ... 543

Глава 12. Генерация кода ... 554
12.0. Введение ... 555
12.1. Генерация определений констант ... 565
12.2. Генерация предложения switch ... 570
12.3. Генерация кода заглушки обработчика сообщения ... 575
12.4. Генерация оберток для данных ... 579
12.5. Генерация функций форматированной печати ... 584
12.6. Генерация Web-клиента для тестирования ввода данных ... 593
12.7. Генерация CGI-сценария для обработки тестовых данных ... 595
12.8. Генерация кода из UML-моделей, описанных на языке XMI ... 601
12.9. Генерация XSLT из XSLT ... 620

Глава 13. Рецепты применения XSLT в вертикальных приложениях 625
13.0. Введение 625
13.1. Преобразование документов из формата Visio VDX в формат SVG 626
13.2. Работа с электронными таблицами в формате Excel XML 641
13.3. Генерация тематических карт из UML-модели с помощью XMI 653
13.4. Генерация Web-сайтов из тематических карт 673
13.5. Генерация документации о SOAP из WSDL-документа 689

Глава 14. Расширение и встраивание XSLT 706
14.0. Введение 706
14.1. Функции расширения в Saxon 707
14.2. Элементы расширения в Saxon 708
14.3. Функции расширения в Xalan-Java 2 709
14.4. Описание Java-функций расширения в формате класса 709
14.5. Описание Java-функций расширения в формате пакета 710
14.6. Описание Java-функций расширения в формате Java 710
14.7. Функции расширения на языке сценариев с использованием встроенного сценария 711
14.8. Элементы расширения в Xalan-Java 2 711
14.9. Реализация элемента расширения на языке Java 711
14.10. Реализация элемента расширения на сценарном языке 712
14.11. Функции расширения в MSXML 713
14.12. Использование встроенных расширений Saxon и Xalan 714
14.13. Расширение XSLT с помощью JavaScript 730
14.14. Реализация функций расширения на языке Java 736
14.15. Реализация элементов расширения на языке Java 744
14.16. Работа с XSLT в программах на языке Perl 760
14.17. Работа с XSLT в программах на языке Java 763

Глава 15. Тестирование и отладка 767
15.0. Введение 767
15.1. Эффективное использование xsl:message 768
15.2. Трассировка потока обработки документа таблицей стилей 771

15.3. Автоматизация вставки отладочной печати 778
15.4. Встраивание тестовых данных в служебные таблицы стилей 786
15.5. Организация автономных тестов 791
15.6. Тестирование граничных условий и ошибочных данных 794

Глава 16. Обобщенное и функциональное программирование 798
16.0. Введение 798
16.1. Создание полиморфного XSLT-кода 805
16.2. Создание обобщенных функций агрегирования элементов 814
16.3. Создание обобщенных функций ограниченного агрегирования 828
16.4. Создание обобщенных функций отображения 836
16.5. Создание обобщенных генераторов наборов узлов 846

Алфавитный указатель 853

Предисловие

XSLT (Extensible Stylesheet Language Transformations) – это технология преобразования XML-документов в другие форматы. Некоторые считают, что освоить ее трудно. Однако тот факт, что в основе технологии лежат шаблоны, делают ее прекрасным кандидатом для изучения на примерах, особенно, если учесть что имеющийся образец зачастую легко приспособить для других целей. Благодаря спецификации XSLT 2.0 выразительность и элегантность XSLT заметно возросли, но язык при этом стал сложнее.

Когда я только начинал работать с XSLT (и позднее, когда приступил к изучению XSLT 2.0), мне очень не хватало сборника рецептов, из которого я мог бы черпать готовые решения стоящих передо мной задач. Первой книгой такого рода, которая попалась мне на глаза, была *Perl Cookbook*, выпущенная издательством O'Reilly. Она заставила меня сначала преодолеть нежелание изучать язык Perl, а затем и полюбить его, в чем оригинальная книга Ларри Уолла (*Programming Perl* – знаменитая «книга с верблюдом») не преуспела. Мне кажется, что такие сборники рецептов нужны, поскольку многим разработчикам недостаточно просто заставить программу работать; они хотели бы в совершенстве овладеть технологией и применять хорошо зарекомендовавшие себя приемы. При этом получить необходимую информацию желательно быстро. А нет лучшего способа глубоко изучить предмет, чем воспользоваться опытом тех, кто уже нашел удачный подход к проблеме.

Острое желание иметь подобную книгу вскоре переросло в желание самому написать ее, тем более что у меня уже подобралось несколько полезных рецептов; что-то придумали другие, а что-то – я сам. Однако я не хотел просто переписывать учебник по XSLT, по-другому сгруппировав материал. Я поставил целью создать полезный ресурс, в котором освещались бы некоторые неочевидные способы применения этой технологии. Заодно я надеялся привлечь внимание тех разработчиков, которые еще не заинтересовались XSLT и тем самым, на мой взгляд, упустили из виду один из наиболее эффективных инструментов работы с XML-документами. Если вы принадлежите к их числу, прошу вас прочитать хотя бы еще несколько абзацев, где я расскажу, чем ценен XSLT и как эта книга поможет вам осознать его потенциал.

XSLT – это язык, который находится одновременно в центре и на обочине магистрального пути развития технологий разработки программного обеспечения. Когда я работал над первым изданием этой книги, мне часто приходилось объяснять знакомым, что такое XSLT и почему так важно тратить время на написание целой книги о нем. Мои знакомые были наслышаны о Java, Perl и даже XML, но ничего не знали об XSLT. Я также видел, как растет потребность в консультациях

по XSLT, в списках рассылки и большем внимании индустрии к этой технологии, которое выражалось бы в книгах, статьях и развитых инструментах поддержки. Хотя число пользователей XSLT возрастает с каждым днем, многие профессионалы все еще не понимают, что это такое и с чем его едят. Я надеюсь, что по мере появления реализаций XSLT 2.0 эта технология распространится шире, однако уверенности в этом нет, в частности из-за конкуренции со стороны XQuery 1.0 и других способов манипулирования XML-документами. Но одно можно сказать точно: изучение XSLT 2.0 не станет потерянным впустую временем, поскольку область его применения будет расширяться, хотя, быть может, и не лавинообразно. К тому же знакомство с XSLT углубит ваши представления о том, как можно обрабатывать XML, даже если в конечном итоге вы остановитесь на другой технологии.

Хотя XSLT 1.0 – вполне зрелый язык и спецификация XSLT 2.0 была принята не так уж давно, у меня есть ощущение, что больше половины людей и компаний, работающих с XML, не используют XSLT. Сравнительно недавно один мой знакомый, который в курсе всех новейших технологий, говорил мне, что XSLT – это просто еще один язык стилизации. Такое недопонимание простительно, поскольку об XSLT судят по первым трем буквам аббревиатуры (XSL – расширенный язык таблиц стилей) и по директиве, с которой начинаются большинство XSLT-программ (xsl:stylesheet). Но самой важной является как раз последняя буква T, означающая *Transformations* (преобразования). Именно это делает язык таким ценным, и по этой причине я им и заинтересовался. При написании этой книги я, в частности, намеревался показать, что XSLT применим к решению самых разных задач. Кроме того, я хотел, чтобы эта книга стала для пользователей начального и среднего уровня тем местом, где можно найти большинство распространенных приемов работы с XSLT. И, наконец, я хотел продемонстрировать, что вообще можно делать с XSLT и тем самым побудить тех, кто уже знает этот язык, к более глубокому изучению, а остальных – присоединиться к компании «преобразователей XML».

Мне доводилось слышать самые разные мнения о том, что такое информатика. Высказывания типа «любое вычисление – это просто более или менее хитроумное манипулирование битами», «компьютеры – не более чем очень усложненные перемалыватели чисел» или «все, что может делать компьютер, возможно выразить в терминах манипулирования символами» в какой-то степени верны. Но я бы хотел предложить и свое собственное обобщение: «Любая задача, решаемая с помощью программы, может быть описана в терминах преобразований». Тот, кто овладел искусством преобразований, овладел и информатикой. Преобразованиями занимается процессор, преобразования лежат в основе алгоритмов, преобразования – это суть работы программистов. И именно преобразования составляют смысл языка XSLT, по крайней мере, когда на вход подается XML-документ (а иногда и нечто иное). Разумеется, XSLT – не единственный язык преобразований, и, как и для тысяч предшествовавших ему языков, неясно, будет ли он развиваться независимо или окажется поглощенной следующей «потрясающей инновацией». Очевидно лишь, что лежащие в основе XSLT идеи никуда не денутся,

поскольку они стары, как и сама информатика. Эта книга поможет читателю понять эти идеи и научиться применять их к конкретным задачам.

О структуре книги

Чтобы сделать книгу полезной максимально широкому кругу читателей, я сохранил большую часть примеров XSLT 1.0 из первого издания. Но добавил и решения на базе XSLT 2.0 в тех случаях, когда они оказывались проще или элегантнее. Иногда я также привожу «2.0-решения» задач, которые было бы очень трудно или даже невозможно решить в рамках версии 1.0. Примеры для версий 1.0 и 2.0 приводятся в разных подразделах. Надеюсь, что так читателям будет проще найти то, что их интересует. Во многих примерах я не даю отдельного решения для версии 2.0. Как правило, это объясняется тем, что, на мой взгляд, «1.0-решение» будет работать и в версии 2.0, а специальная адаптация под 2.0 мало что дает. Искренне надеюсь, что мое желание сохранить деревья и сэкономить время не вызовет раздражения у читателя.

Версии XSLT 1.0 и 2.0 опираются на фундамент, заложенный спецификациями XPath 1.0 и 2.0 соответственно. Некоторые читатели первого издания пеняли мне на то, что я не уделил должного внимания XPath. Глава 1 написана отчасти для того, чтобы удовлетворить их, а отчасти потому, что язык XPath 2.0 заметно расширился и усложнился.

Одно из самых простых преобразований – это обработка последовательности символов, то есть *строки*. В отличие от древнего языка SNOBOL и относительно современного Perl, XSLT не проектировался специально для манипулирования строками. Однако в главе 2 показано, что практически все, что необходимо при работе со строками, можно выполнить, оставаясь в рамках XSLT, причем новые функции, появившиеся в XSLT 2.0, значительно упрощают задачу.

Еще один вид низкоуровневых преобразований – это численные преобразования (собственно, это не что иное, как *математика*). Они пронизывают программирование снизу доверху просто потому, что измерения и расчеты – неотъемлемая часть самой жизни. В главе 3 показано, как обогатить «математические способности» XSLT, хотя этот язык и не задумывался как полноценная замена Фортрану.

Манипулирование датами и временем заложено в самой природе человека, интерес к часам, календарям и точному прогнозированию часто становился движущей силой прогресса. Глава 4 содержит рецепты работы с датами и временем, которые восполняют недостатки, присущие стандартному XSLT 1.0. Там же описываются долгожданные функции, которые наконец-то были добавлены в XSLT 2.0. Рассматривая в этой главе интригующие и трудные задачи, связанные с преобразованием дат, мы приведем как готовые решения, так и ссылки на ресурсы, относящиеся к летоисчислению.

Для любого преобразования нужно прежде всего указать объект, к которому оно применяется. Если этот объект составной, то в ходе преобразования нужно будет обойти его части. Этой теме посвящена глава 5, в которой исследуются

проблемы, для решения которых и предназначен XSLT. Здесь мы представляем XML-документ в виде дерева и показываем, как XSLT позволяет манипулировать такими деревьями. Попутно приводятся советы по поводу достижения максимальной производительности при обработке XML-документов.

Глава 6 была заново написана для второго издания. Она целиком посвящена версии XSLT 2.0. Читателям, которых больше всего интересуют новшества, появившиеся в этой версии, рекомендуется сначала прочесть главы 1 и 6, а затем для более углубленного понимания познакомиться с примерами ее применения, рассыпанными по всему тексту.

Задолго до появления различных текстовых процессоров, HTML и PDF существовал старый добрый текст. Весьма важна задача преобразования данных из формата, предназначенного для обработки компьютером, к виду, удобному человеку. Если исходные данные представлены в формате XML, то для ее решения XSLT подходит идеально. В главе 7 демонстрируется, как из XML-документа можно извлечь текст, пригодный для вывода на терминал, подачи на вход текстового редактора или для импорта в программы, требующие, чтобы значения были как-то отделены друг от друга, например, запятыми.

XML быстро становится универсальным форматом для передачи информации, и есть все признаки того, что эта тенденция будет и дальше набирать силу, а не сойдет на нет. Поэтому очень часто одни XML-документы приходится преобразовывать в другие. В главе 8 рассматриваются именно такие преобразования. Здесь показано, как XML-документы можно разбивать на части, объединять, преобразовывать из иерархического вида в плоский, очищать и выполнять иные трансформации с помощью сравнительного небольших XSLT-шаблонов.

Многие преобразования сводятся просто к извлечению информации из имеющихся данных для получения ответа на тот или иной вопрос. Глава 9 – это сокровищница рецептов, иллюстрирующих применение XSLT в качестве языка запросов. В ней приведено немало примеров, моделирующих запросы, которые вполне могут возникнуть и у вас.

Еще одна важная сфера применения XSLT – это HTML-разметка. В главе 10 показано, как решать задачи генерирования Web-контента, в том числе ссылок, таблиц, фреймов, форм и выполнять другие преобразования на стороне клиента.

Задача графического программирования – преобразование данных к визуально воспринимаемому виду. Не стоит думать, что XSLT – это язык графического программирования. Однако, если нужно выполнить преобразование в формат SVG (Scalable Vector Graphics), то XSLT поможет добиться впечатляющих результатов. В главе 11 демонстрируется, как преобразовать исходные данные в столбчатые или секторные диаграммы, линейные графики и другие графические компоненты. Здесь же рассматриваются преобразования XML в древовидное представление. В этой главе подчеркивается, что преобразования – это конструкции, которые можно комбинировать для получения самых разных результатов на выходе.

Меня всегда интересовала задача автоматической генерации кода. С этим преобразованием люди пока справляются лучше компьютеров (к счастью для тех, кто зарабатывает на жизнь программированием). Однако иногда лучше написать

программу, которая будет генерировать код, чем писать этот код самому. В главе 12 описываются плюсы представления данных, управляющих генерацией кода, в виде XML и доказывается, что XSLT идеально подходит для написания генераторов кода на языках C++, Java и самом XSLT. Сюда же включен рецепт генерации кода, заимствованный из паттерна проектирования, который представлен на языке UML, записанном в XML-нотации.

Язык XSLT применим и в более сложных ситуациях. В главе 13 приведены примеры такого рода, в том числе преобразование из формата Visio VDX в SVG, преобразование Microsoft Excel XML, построение тематических карт и обработка WSDL-документов.

Язык XSLT и сам по себе достаточно мощный, но его можно расширять и встраивать в другие языки, что позволяет делать поистине поразительные вещи. В главе 14 обсуждаются расширения XSLT на примере языков Java и JavaScript. Здесь же показано, как XSLT можно встраивать в программы на языках Java и Perl.

Тестирование и отладка – неотъемлемые составные части разработки на любом языке, и XSLT – не исключение. В главе 15 описываются некоторые полезные приемы поиска ошибок в XSLT-программах, которые дают результат даже в отсутствие настоящего отладчика XSLT.

В главе 16 еще раз подчеркивается мысль, что XSLT – гораздо больше, чем просто очередной язык стилизации. Основное внимание уделено использованию XSLT в качестве языка обобщенного и функционального программирования. Даже если вы не найдете в этой главе конкретных полезных рецептов, она откроет вам глаза на то, как можно использовать XSLT для создания обобщенных решений, и подтолкнет к размышлениям в этом направлении.

Типографские соглашения

В книге применяются следующие шрифты:
Курсив:

- пути, имена файлов и программ;
- адреса в Интернете, например, имена доменов и URL;
- новые термины при первом употреблении.

`Моноширинный`:

- команды и параметры, которые следует вводить буквально;
- имена и ключевые слова в программах, в том числе имена методов, переменных и классов;
- теги элементов XML.

`Моноширинный полужирный` – для выделения участка кода.
`Моноширинный курсив` – для переменных аргументов программы.

Этим значком обозначается примечание к окружающему тексту.

Этим значком обозначается предупреждение, относящееся к окружающему тексту.

О примерах кода

Эта книга призвана помогать вам в работе. Поэтому вы можете использовать приведенный в ней код в собственных программах и в документации. Спрашивать у нас разрешения необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, никто не возбраняет включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров на компакт-диске разрешение требуется. Цитировать книгу и примеры в ответах на вопросы можно без ограничений. Но для включения значительных объемов кода в документацию по собственному продукту нужно получить разрешение.

Мы высоко ценим, хотя и не требуем, ссылки на наши издания. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «*XSLT Cookbook*, Second Edition by Sal Mangano, Copyright 2006 O'Reilly Media, Inc., 0-596-00974-7.»

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу *permissions@oreilly.com*.

Для этой книги организована сопроводительная Web-страница, где публикуются сведения об ошибках, примеры и вообще дополнительная информация. Ее адрес в Интернете:

http://www.oreilly.com/catalog/xsltckbk

Замечания и технические вопросы можно направлять по адресу:

bookquestions@oreilly.com

Дополнительная информация о книгах, различные конференции, ресурсные центры и сеть O'Reilly Network можно найти на сайте:

http://www.oreilly.com/

Благодарности

Благодарности ко второму изданию

Писать книгу с нуля, – безусловно, тяжелая задача, особенно если это первая книга автора. Но, как оказалось, работа над вторым изданием по разным причинам была не менее сложной и интересной. Завершить ее не удалось бы без поддержки коллег, друзей и семьи. Кроме того, ни возможности, ни побудительных причин для второго издания не было бы, если бы не всесторонняя помощь, добрые слова и доброжелательная критика со стороны читателей первого издания.

Должен еще раз поблагодарить своего редактора Саймона Сен-Лорана (Simon St.Laurant) за его поистине безграничное терпение вопреки многократно срываемым мной срокам, а также за мудрые советы в большом и в малом.

Второе издание не состоялось бы без титанических усилий Майкла Кэя (Michael Kay), который не только отредактировал черновики материалов, посвященных XPath 2.0 и XSLT 2.0, но – и это особенно важно – предоставил бесплатную высококачественную реализацию того и другого в Saxon 8.

Должен также выразить признательность Эвану Ленцу (Evan Lenz) и Майку Фицджеральду (Mike Fitzgerald) за совместную работу по техническому редактированию, которая принесла великолепные плоды. Если читатель сочтет, что пояснения технически недостаточно ясны, неточны или вообще неправильны, то это лишь потому, что ваш покорный слуга упрямо проигнорировал или недопонял их предложения.

Опыт работы с XPath 2.0 и XSLT 2.0 я приобрел в основном в ходе разработки сайта SD Times (www.sdtimes.com). Хотелось бы поблагодарить Тэда Бара (Ted Bahr) и Алана Цейчика (Alan Zeichick) из компании BZ Media за предоставленную возможность переработать их сайт, а также Ребекку Паппас (Rebecca Pappas), которая терпеливо мирилась с тем, что я вынужден делить время между ней и книгой. Благодарю также своих клиентов и друзей из компании SIAC, особенно Кэрол Спивак (Carol Spiewak), Фрэнка Карреру (Frank Carrera), Берта Спильмана (Bert Spielman), Ами Ху (Amy Hui) и Диану Веркавиц (Diana Verkavits), которые нанимали меня для решения многих интересных задач, в ходе работы над которыми я отточил свое мастерство разработчика программ.

Наконец, я хочу сказать спасибо своей жене Ванде и сыновьям Ленардо и Сальваторе за то, что они пережили еще одну мою книгу. Я знаю, что совсем невесело смотреть, как папа сидит за компьютером, особенно когда на улице столько всего интересного и неожиданного. Впрочем, надеюсь, вы поймете, что упорный труд и стремление к цели вознаграждаются, если, конечно, время от времени делать небольшие перерывы, чтобы побездельничать (и пошутить насчет треснувших и разбитых предметов!).

Благодарности к первому изданию

Я всегда мечтал написать книгу и очень рад, что издательство O'Reilly помогло мне реализовать свою мечту. Однако это вовсе не было подвигом одиночки. На пути к цели мне помогали многие люди, и я хотел бы отметить их заслуги.

Прежде всего, благодарю своего редактора Саймона Сен-Лорана. Он сопровождал меня на всем пути, начиная с момента получения написанного в спешке письма с предложением о сотрудничестве и до последних этапов производственного цикла. Саймон всегда готов был поддержать меня, разделить мои радости и разочарования, без которых не обходится ни одно творчекое начинание.

Во-вторых, я выражаю признательность Джени Теннисон (Jeni Tennison), моему первому техническому редактору. Ее технические знания и внимание к деталям просто бесподобны. Джени не только исправила мои откровенные и менее очевидные ошибки, но и любезно позволила включить в книгу свой код и воспользоваться идеями, которыми она щедро делится во многих посвященных XML форумах, в которых участвует. (Если остались какие-то ошибки, то только из-за моей собственной бестолковости.) Джени – просто уникум, и я уверен, что сообщество пользователей XML присоединится к моим благодарностям за ее вклад и бескорыстную помощь.

В-третьих, я хотел бы поблагодарить всех моих коллег по работе в банке Морган Стэнли за поддержку и одобрение моей работы, а особенно своего начальника

Фарида Халили (Farid Khalili) за то, что он с пониманием относился к тем случаям, когда мне нужно было срываться с работы или оставаться дома, чтобы успеть к сроку. А также его начальника Джона Рейнольдса (John Reynolds), который рекомендовал мою книгу всему возглавляемому им отделению бумаг с фиксированным доходом. Также хочу сказать спасибо моему бывшему клиенту, корпорации SIAC, и особенно Карен Халберт (Karen Halbert), позволившей мне возглавить проект, на котором я отточил навыки работы с XSLT.

В-четвертых, спасибо всем, кто любезно предоставил мне материалы для этой книги, в том числе: Стиву Боллу (Steve Ball), Джону Брину (John Breen), Джейсону Даймонду (Jason Diamond), Никите Огивецкому и Джени Теннисон. Спасибо также пришедшим Джени на смену техническим редакторам Мике Дубинко (Micah Dubinko) и Джирке Косеку (Jirka Kosek), чьи замечания и предложения оказались весьма ценными. Ну и, конечно, всему коллективу производственного отдела издательства O'Reilly, без которого эта работа не вышла бы в свет.

И, наконец, хочу выразить признательность своим родителям, семье и друзьям. Вы, как обычно, всемерно поддерживали и подпитывали меня, помогая вести сбалансированную жизнь. Больше всего я благодарен своей жене Ванде и сыну Леонардо, без их моральной поддержки и неисчислимых жертв эта книга не состоялась бы. Спасибо тебе, Ванда, за все то, что должен был бы сделать я, а пришлось делать тебе, пока я томился как раб в темнице! Спасибо тебе, Леонардо, за то, что ты говорил: «Папа, иди работай», когда хотелось сказать: «Папа, давай поиграем». Вы вдвоем – мое самое большое достижение в жизни.

Глава 1. Язык XPath

> Нео, рано или поздно ты, как и я, поймешь, что есть разница между осознанием пути и следованием по нему.
>
> *Морфеус (Матрица)*

XPath – это язык для записи выражений. Он имеет фундаментальное значение для обработки XML-документов. Нельзя овладеть XSLT, не зная XPath, точно так же, как нельзя выучить английский язык, не зная алфавита. Некоторые читатели первого издания этой книги пеняли мне за то, что я не включил в нее основ XPath. Эта глава добавлена отчасти, чтобы удовлетворить их, но главным образом потому, что в спецификации XPath 2.0 выразительная мощь этого языка была значительно усилена. Впрочем, многие рецепты будут работать и с XPath 1.0.

В XSLT 1.0 язык XPath используется тремя способами. Во-первых, в шаблонах он служит для адресации частей преобразуемого документа. Во-вторых, он применяется для задания образцов в правилах сопоставления. В-третьих, с помощью встроенных в XPath операторов и функций выполняются простые математические операции и манипуляции со строками.

В XSLT 2.0 связь с XPath 2.0 сохранена и даже стала прочнее. В нем широко используются новые вычислительные средства, появившиеся в XPath 2.0. Можно даже сказать, что дополнительные возможности XSLT 2.0 – во многом результат нововведений в XPath 2.0, к числу которых относятся последовательности, регулярные выражения, условные и итеративные выражения, система типов, совместимая со спецификацией XML Schema, а также множество новых встроенных функций.

Каждый рецепт в этой главе – это подборка мини-рецептов для решения определенного класса задач, возникающих в XPath в контексте XSLT. Все XPath-выражения прокомментированы в соответствии с принятым в XPath 2.0 соглашением `(: комментарий :)`, но пользователям XPath 1.0 следует иметь в виду, что это недопустимая синтаксическая конструкция. Пустой результат вычисления XPath-выражения мы будем обозначать `()`, именно так в XPath 2.0 записывается пустая последовательность.

1.1. Применение осей

Задача

Требуется отобрать узлы XML-дерева с учетом сложных взаимосвязей в иерархической структуре.

Решение

Во всех приведенных ниже примерах используются оси. В каждой группе для демонстрации берется некий XML-документ, в котором контекстный узел выделен полужирным шрифтом. Поясняется, что является результатом вычисления пути, при этом показано, какие элементы отбираются относительно выделенного контекста. В некоторых случаях для иллюстрации тонкостей вычисления конкретного выражения рассматриваются и другие узлы, помимо контекстного.

Дочерняя ось и ось потомков

Дочерняя ось принимается в XPath по умолчанию. Иными словами, явно указывать ось `child::` необязательно, но, если вы хотите быть педантом, то можете и указать. Спуститься по XML-дереву глубже, чем на один уровень, позволяют оси `descendant::` и `descendant-or-self::`. Первая не включает сам контекстный узел, вторая – включает.

```
<Test id="descendants">
    <parent>
            <X id="1"/>
            <X id="2"/>
            <Y id="3">
                    <X id="3-1"/>
                    <Y id="3-2"/>
                    <X id="3-3"/>
            </Y>
            <X id="4"/>
            <Y id="5"/>
            <Z id="6"/>
            <X id="7"/>
            <X id="8"/>
            <Y id="9"/>
    </parent>
<Test>

(: Отобрать все дочерние элементы с именем X :)
X   (: то же, что child::X)

Результат: <X id="1"/> <X id="2"/> <X id="4"/> <X id="7"/> <X id="8"/>
```

```
(: Отобрать первый дочерний элемент с именем X :)
X[1]
Результат: <X id="1"/>

(: Отобрать последний дочерний элемент с именем X :)
X[last()]
Результат: <X id="8"/>

(: Отобрать первый дочерний элемент при условии, что его имя X. Иначе
пусто :)
*[1][self::X]

Результат: <X id="1"/>

(: Отобрать последний дочерний элемент при условии, что его имя X. Иначе
пусто :)
*[last()][self::X]

Результат: ()

(: Отобрать последний дочерний элемент при условии, что его имя Y. Иначе
пусто :)
*[last()][self::Y]

Результат: <Y id="9"/>

(: Отобрать всех потомков с именем X :)
descendant::X

Результат: <X id="1"/> <X id="2"/> <X id="3-1"/> <X id="3-3"/>
<X id="4"/> <X id="7"/> <X id="8"/>

(: Отобрать контекстный узел, если его имя X, а также всех потомков с
именем X :)
descendant-or-self::X

Результат: <X id="1"/> <X id="2"/> <X id="3-1"/> <X id="3-3"/>
<X id="4"/> <X id="7"/> <X id="8"/>

(: Отобрать контекстный узел и всех его потомков :)
descendant-or-self::*
Результат: <parent> <X id="1"/> <X id="2"/> <Y id="3"> <X id="3-1"/>
<Y id="3-2"/> <X id="3-3"/> </Y> <X id="4"/> <Y id="5"/> <Z id="6"/>
<X id="7"/> <X id="8"/> <Y id="9"/> </parent>
```

Оси братьев

Оси братьев называются `preceding-sibling::` и `following-sibling::`. Ось `preceding-sibling` содержит братьев, предшествующих контекстному узлу, а ось `following-sibling` – следующих за ним. Братьями, естественно, называются дети одного родителя. Почти во всех примерах ниже используется ось `preceding-sibling::`, но вам не составит труда вычислить результат и для оси `following-sibling::`.

Имейте в виду, что в позиционном выражении вида `preceding-sibling::*[1]` вы ссылаетесь на непосредственно предшествующего брата в порядке обратного отсчета от контекстного узла, а не первого брата в порядке документа. Некоторых смущает тот факт, что результирующая последовательность возвращается в порядке документа вне зависимости от того, используется ось `preceding-sibling::` или `following-sibling::`. В выражении `../X`, строго говоря, никакая ось не указана; это просто способ отобрать предшествующего и последующего брата с именем X, а также сам контекстный узел, если он называется X. Формально это сокращенная запись выражения `parent::node()/X`. Отметим, что выражения `(preceding-sibling::*)[1]` и `(following-sibling::*)[1]` отбирают первого предшествующего или последующего брата в порядке документа.

```
<!-- Тестовый документ с выделенным контекстным узлом ->
<Test id="preceding-siblings">
    <A id="1"/>
    <A id="2"/>
    <B id="3"/>
    <A id="4"/>
    <B id="5"/>
    <C id="6"/>
    <A id="7"/>
    <A id="8"/>
    <B id="9"/>
</Test>

(: Отобрать всех братьев с именем A, предшествующих контекстному узлу :)
preceding-sibling::A

Результат: <A id="1"/> <A id="2"/> <A id="4"/>

(: Отобрать всех братьев с именем A, следующих за контекстным узлом :)
following-sibling::A

Результат: <A id="8"/>
(: Отобрать всех братьев, предшествующих контекстному узлу :)
preceding-sibling::*

Результат: <A id="1"/> <A id="2"/> <B id="3"/> <A id="4"/> <B id="5"/>
```

```
<C id="6"/>

(: Отобрать первого предшествующего брата с именем A в обратном порядке
документа :)
preceding-sibling::A[1]

Результат: <A id="4"/>

(: Отобрать первого предшествующего брата в обратном порядке документа
при условии, что его имя A :)
preceding-sibling::*[1][self::A]

Результат: ()
(: Если бы контекстным был узел <A id="8"/>, то в результате мы получили
бы <A id="7"/> :)

(: Отобрать всех предшествующих братьев, кроме элементов с именем A :)
preceding-sibling::*[not(self::A)]

Результат: <B id="3"/> <B id="5"/> <C id="6"/>

(: В следующих примерах используется такой тестовый документ :)

<Test id="preceding-siblings">
    <A id="1">
           <A/>
    </A>
    <A id="2"/>
    <B id="3">
           <A/>
    </B>
    <A id="4"/>
    <B id="5"/>
    <C id="6"/>
    <A id="7"/>
    <A id="8"/>
    <B id="9"/>
</Test>
(: Элемент, непосредственно предшествующий контекстному узлу при усло-
вии, что у того есть дочерний элемент
с именем A :)
preceding-sibling::*[1][A]
Результат: ()
(: Первый из предшествующих контекстному узлу элементов,
у которого есть дочерний элемент с именем A :)
```

```
preceding-sibling::*[A][1]

Результат: <B id="3"/> ...

(: XPath 2.0 позволяет более гибко выбирать элементы с учетом про-
странств имен. В следующих примерах используется такой XML-документ :)

<Test xmlns:NS="http://www.ora.com/xsltckbk/1" xmlns:NS2= "http://
www.ora.com/xsltckbk/2">
    <NS:A id="1"/>
    <NS2:A id="2"/>
    <NS:B id="3"/>
    <NS2:B id="3"/>
</Test>
(: Отобрать всех предшествующих братьев контекстного узла, для которых
пространство имен ассоциировано с префиксом NS :)
preceding-sibling::NS:*

Результат: <NS:A id="1"/>

(: Отобрать всех предшествующих братьев контекстного узла, для которых
локальное имя равно A :)
preceding-sibling::*:A

Результат: <NS:A id="1"/> <NS2:A id="2"/>
```

Родительская ось и ось предков

Родительская ось (`parent::`) относится к родителю контекстного узла. Не путайте выражение `parent::X` с `../X`. Первое порождает последовательность, которая содержит в точности один элемент, если имя родителя контекстного узла – X, и пуста в противном случае. Второе – это сокращенная запись выражения `parent::node()/X`, которое отбирает всех братьев контекстного узла с именем X, в том числе и сам этот узел, если он называется X.

С помощью осей `ancestor::` и `ancestor-or-self::` можно перемещаться вверх по XML-дереву (переходя к родителю, деду, прадеду и т.д.). В первом случае сам контекстный узел исключается, во втором – включается.

```
(: Возвращает родителя контекстного узла, при условии, что он называется
X, в противном случае — пустую последовательность. :)
parent::X
(: Возвращает родителя контекстного узла. Результат может быть пустым,
только если контекстный узел является элементом верхнего уровня. :)
parent::*

(: Возвращает родителя, если его пространство имен ассоциировано с
префиксом X. Префикс должен быть определен, иначе произойдет ошибка. :)
```

```
parent::NS:*

(: Возвращает родителя независимо от его пространства имен при условии,
что локальное имя равно X.:)
parent::*:X

(: Возвращает всех предков (включая родителя) с именем X. :)
ancestor::X

(: Возвращает контекстный узел, если он называется X, а также всех
предков с именем X. :)
ancestor-or-self::X
```

Оси предшественников и последователей

Оси `preceding::` и `following::` могут отбирать очень много узлов, поскольку учитываются все узлы, предшествующие контекстному (или следующие за ним) в порядке документа, исключая предков в случае оси `following::` или потомков для оси `preceding::`. Не забывайте также, что обе оси не включают узлы, соответствующие пространствам имен и атрибутам.

```
(: Все предшествующие узлы с именем X. :)
preceding::X

(: Ближайший предшественник с именем X. :)
preceding::X[1]

(: Самый дальний последователь с именем X. :)
following::X[last()]
```

Обсуждение

В языке XPath понятие оси служит для того, чтобы выделить в дереве документа различные подмножества узлов относительно некоторого узла, называемого контекстным. В общем случае эти подмножества пересекаются, но оси ancestor, descendant, following, preceding и self разбивают документ на непересекающиеся части, в совокупности содержащие все узлы (за исключением тех, что соответствуют пространствам имен и атрибутам). Контекстный узел устанавливается языком, в который погружен XPath. В случае XSLT контекстный узел устанавливают следующие конструкции:

- сопоставление с шаблоном (`<xsl:template match="x">…</xsl:template>`);
- `xsl:for-each`;
- `xsl:apply-templates`.

Свободное владение языком написания путевых выражений – необходимое условие для выполнения как простых, так и сложных преобразований. Опыт программирования на традиционных языках часто служит причиной путаницы

и ошибок при работе с XPath. Например, я часто ловил себя на том, что писал что-то типа `<xsl:if test="preceding-sibling::X[1]"> </xsl:if>`, имея в виду `<xsl:if test="preceding-sibling::*[1][self::X]"></xsl:if>`. Возможно, это объясняется тем, что последнее выражение – интуитивно менее понятный способ выразить такое условие: «если имя непосредственно предшествующего брата равно X».

Конечно, нет никакой возможности показать все полезные комбинации осей в путевых выражениях. Но, если вы поймете приведенные выше примеры, то сможете разобраться, что означает выражение `preceding-sibling::X[1]/descendant::Z[A/B]` и еще более сложные.

1.2. Фильтрация узлов

Задача

Требуется отобрать узлы в зависимости от хранящихся в них данных, возможно, в сочетании с ограничениями на имена или позицию.

Решение

Во многих мини-рецептах в разделе 1.1 для фильтрации узлов использовались предикаты, но они налагали условия только на позицию или имя узла. Ниже мы рассмотрим предикаты, фильтрующие по содержимому. Во всех примерах из этого раздела перед каждым предикатом стоит имя X, но вместо него можно было бы задать произвольное путевое выражение, в частности, любое из рассмотренных в разделе 1.1.

В примерах ниже употребляются операторы сравнения из XPath 2.0 (`eq`, `ne`, `lt`, `le`, `gt` и `ge`) вместо привычных операторов `=`, `!=`, `<`, `<=`, `>`, `>=`. Объясняется это тем, что для сравнения атомарных значений новые операторы предпочтительнее. В XPath 1.0 есть только старые операторы, так что вам придется внести коррективы. Новые операторы были введены в XPath 2.0, так как обладают более простой семантикой и потому могут оказаться эффективнее. Недостатки старых операторов проявляются, когда по обе стороны сравнения оказываются последовательности. В рецепте 1.8 эта тема рассматривается более подробно.

Для тех, кто работает с XPath 2.0, будет уместно сделать еще одно замечание. В этой версии используется информация о типах, если доступна схема. В некоторых из приведенных ниже выражений это может стать причиной ошибок типизации. Например, `X[@a = 10]` и `X[@a = '10']` – не одно и то же, если атрибут `a` имеет целочисленный тип. Мы предполагаем, что схема не задана, и потому все атомарные значения имеют тип `untypedAtomic`. Дополнительную информацию по этому поводу см. в рецептах 1.9 и 1.10.

```
(: Отобрать детей элемента X, у которых есть атрибут a. :)
X[@a]
```

```
(: Отобрать детей элемента X, у которых есть хотя бы один атрибут. :)
X[@*]

(: Отобрать детей элемента X, у которых есть хотя бы три атрибута. :)
X[count(@*) > 2]

(: Отобрать детей элемента X, для которых сумма всех атрибутов меньше 7. :)
X[sum(foreach $a in @* return number($a)) < 7]
(: В XSLT 1.0 надо писать sum(@*) &lt; 7 :)

(: Отобрать детей элемента X, у которых нет атрибутов. :)
X[not(@*)]

(: Отобрать детей элемента X, у которых есть атрибут a со значением '10'. :)
X[@a eq '10']

(: Отобрать детей элемента X, у которых есть дочерний элемент Z со
значением '10'. :)
X[Z eq '10']

(: Отобрать детей элемента X, у которых есть дочерний элемент Z со
значением, отличным от '10'. :)
X[Z ne '10']

(: Отобрать детей элемента X, у которых есть хотя бы один текстовый
дочерний элемент. :)
X[text()]

(: Отобрать детей элемента X, у которых есть текстовый дочерний элемент,
содержащий хотя бы один непробельный символ. :)
X[text()][normalize-space(.)]
(: Отобрать детей элемента X, у которых есть хотя бы один дочерний
элемент. :)
X[node()]

(: Отобрать тех детей элемента X, которые содержат комментарий. :)
X[comment()]

(: Отобрать детей элемента X, у которых атрибут @a имеет числовое значение,
меньшее 10. Это выражение одинаково хорошо работает в XPath 1.0 и XPath 2.0
независимо от того, имеет ли атрибут @a числовой или строковый тип. :)
X[number(@a) < 10]

(: Отобрать элемент X, если у него есть хотя бы один предшествующий брат
```

```
с именем Z, который содержит атрибут y, не равный 10. :)
X[preceding-sibling::Z/@y ne '10']

(: Отобрать детей элемента X, строковое значение которых состоит из
единственного пробела. :)
X[. = ' ']

(: Странный способ получить пустую последовательность! :)
X[false()]

(: То же, что X. :)
X[true()]

(: Элементы X, имеющие ровно 5 дочерних элементов. :)
X[count(*) eq 5]

(: Элементы X, имеющие ровно 5 дочерних узлов (включая элементы, текст,
комментарии и узлы с командами обработки, но исключая узлы-атрибуты). :)
X[count(node()) eq 5]

(: Элементы X, имеющие ровно 5 дочерних узлов любого вида. :)
X[count(@*) | node()) eq 5]

(: Первый дочерний элемент X при условии, что его значение равно 'some
text', в противном случае пустая последовательность. :)
X[1][. eq 'some text']

(: Отобрать всех детей элемента X со значением 'some text' и вернуть
первый из них либо пустую последовательность, если таковых не существует.
Проще говоря, вернуть первый дочерний элемент X, имеющий строковое
значение 'some text'. :)
X[. eq 'some text'][1]
```

Обсуждение

Как и в рецепте 1.1, невозможно рассмотреть все интересные комбинации фильтрующих предикатов. Однако, разобравшись с приведенными примерами, вы сможете написать практически любой нужный вам предикат. Отметим также, что с помощью логических операторов `and`, `or` и функции `not()` можно составлять и более сложные условия:

```
number(@a) > 5 and X[number(@a) < 10]
```

Применяя предикаты со сложными выражениями, не забывайте о скобках.

```
(: Отобрать первый дочерний элемент с именем Y для каждого дочернего
```

```
элемента X контекстного узла. Это выражение может вернуть последователь-
ность, содержащую несколько элементов Y. :)
X/Y[1]

(: Отобрать последовательность узлов X/Y и вернуть первый из них.
Это выражение может вернуть не более одного элемента Y. :)
(X/Y)[1]
```

Ученый-теоретик сказал бы, что оператор [] имеет более высокий приоритет, чем оператор /.

1.3. Работа с последовательностями

Задача

Вы хотите манипулировать наборами произвольных узлов и атомарных значений, взятых из одного или нескольких XML-документов.

Решение

XPath 1.0

В версии XPath 1.0 нет понятия последовательности, а потому приведенные ниже рецепты к ней как правило неприменимы. В XPath 1.0 есть наборы узлов. Вот идиоматический способ сконструировать пустой набор узлов в XPath 1.0:

```
(: Пустой набор узлов :)
/..
```

XPath 2.0

```
(: Конструктор пустой последовательности. :)
()

(: Последовательность, состоящая из единственного атомарного элемента со
значением 1. :)
1

(: Сконструировать последовательность с помощью оператора "запятая".
Здесь мы строим последовательность, в которую входят все дочерние эле-
менты контекстного узла с именем X, затем элементы с именем Y и затем
элементы с именем Z. :)
X, Y, Z

(: Применение оператора to для конструирования диапазона. :)
1 to 10

(: Сочетание оператора "запятая" и нескольких диапазонов. :)
```

```
1 to 10, 100 to 110, 17, 19, 23

(: Можно включать также переменные и функции. :)
1 to $x
1 to count(para)

(: Последовательности не бывают вложенными, поэтому следующие два
примера дают одинаковый результат. :)
((1,2,3), (4,5,(6,7),8,9,10))
1,2,3,4,5,6,7,8,9,10

(: Оператор to не может создавать убывающие диапазоны. :)
10 to 1 (: Эта последовательность пуста! :)

(: Но нужного результата можно достичь таким способом. :)
for $n in 1 to 10 return 11 - $n

(: Удалить из последовательности дубликаты. :)
distinct-values($seq)

(: Вернуть длину последовательности. :)
count($seq)
(: Проверить, пуста ли последовательность. :)
empty($seq)(: это лучше, чем count($seq) eq 0 :)

(: Найти, в каких позициях последовательности находится данный элемент.
Функция index-of возвращает последовательность целых чисел, соответствующих позициям тех элементов последовательности, заданной первым аргументом, которые имеют значение, равное второму аргументу. :)
index-of($seq, $item)

(: Выделение подпоследовательностей. :)

(: Не более трех элементов $seq, начиная со второго. :)
subsequence($seq, 2, 3)

(: Элементы $seq, начиная с третьего и до конца. :)
subsequence($seq, 3)

(: Вставить последовательность $seq2 перед третьим элементом последовательности $seq1. :)
insert-before($seq1, 3, $seq2)

(: Построить новую последовательность, которая содержит все элементы
```

```
$seq, кроме третьего. :)
remove($seq, 3)

(: Для удаления нескольких элементов можно воспользоваться выражениями
такого вида. :)
$seq1[not(position() = (1,3,5))]
$seq1[position() gt 3 and position() lt 7]
```

Обсуждение

В XPath 2.0 каждый элемент данных (значение) является последовательностью. Таким образом, атомарное значение 1 – последовательность, как и результат вычисления выражения (1 to 10). По-другому эту мысль можно выразить, сказав, что вычисление любого выражения в XPath 2.0 дает последовательность. Последовательность может содержать нуль и более элементов, в качестве которых могут выступать узлы и атомарные значения в любом сочетании. При сравнении последовательностей порядок существенен. Нумерация элементов начинается с 1 (а не с 0, как в языках C и Java).

В XPath 1.0 нет последовательностей, но есть наборы узлов. Набор узлов не так удобен, но во многих случаях различие несущественно. Например, любое выражение XPath 1.0, в котором используются функции count() и empty(), ведет себя так же, как в версии 2.0. Преимущество XPath 2.0 заключается в том, что последовательности – это полноценные конструкции, для создания и манипулирования которыми имеются многочисленные функции, включенные в версию 2.0. В рецептах из этого раздела описываются многие важные идиомы для работы с последовательностями. Дополнительные примеры вы встретите и в других разделах.

1.4. Включение условий в выражения if

Задача

У вас есть сложная XSLT-программа, которая оказалась излишне длинной из-за того, что запись условий if-then-else в XML слишком многословна.

Решение

XPath 1.0

В XPath 1.0 есть несколько приемов, позволяющих обойтись без громоздкого выражения `xsl:choose` в простых ситуациях. Они основаны на том, что в числовом контексте значение false преобразуется в 0, а значение true – в 1.

Так, например, минимум, максимум и абсолютную величину можно вычислить непосредственно. Ниже предполагается, что `$x` и `$y` – целые числа.

```
(: min :)
($x <= $y) * $x + ($y < $x) * $y
```

```
(: max :)
($x >= $y) * $x + ($y > $x) * $y

(: abs :)
(1 - 2 * ($x < 0)) * $x
```

XPath 2.0

Для описанных в предыдущем разделе операций (min, max, abs) в XPath 2.0 появились специальные функции. Чтобы реализовать другие простые условия, можно воспользоваться новой конструкцией – выражением `if`.

```
(: Вместо отсутствующего атрибута взять значение по умолчанию 10. :)
if (@x) then @x else 10

(: Вместо отсутствующего элемента взять значение по умолчанию
'unauthorized'. :)
if (password ) then password else 'unauthorized'

(: Защита от деления на нуль. :)
if ($d ne 0) then $x div $d else 0

(: Текст элемента para, если он содержит хотя бы один непробельный
символ, иначе единственный пробел. :)
if (normalize-space(para)) then string(para) else ' '
```

Обсуждение

Если вы давно работаете с XSLT 1.0, то, наверное, кривитесь всякий раз, как нужно включить в шаблон условно выполняемый код, По себе знаю, часто приходилось применять встроенные в XSLT средства сопоставления с образцом, лишь бы обойтись без условий. И дело не в том, что такой код сложнее или менее эффективен, просто он получается очень многословным. Простое условие `xsl:if` еще терпимо, но, если дело доходит до ветвления if-then-else, то приходится обращаться к громоздкой конструкции `xsl:choose`. Печально это.

В XSLT 2.0 появилась альтернатива, но она является частью XPath 2.0, а не самого языка XSLT. При первом знакомстве складывается впечатление, что XPath только испортили введением конструкции, которая применяется для управления логикой выполнения программы. Однако, начав активно использовать XPath 2.0, вы скоро поймете, что и XPath, и XSLT только выиграли от такого решения. К тому же условные выражения XPath 2.0 не отменяют конструкции `xsl:if`, а лишь уменьшают потребность в ней в тех ситуациях, когда она выглядит особенно неуклюже. В качестве иллюстрации рассмотрим такой фрагмент:

```
<!-- XSLT 1.0 -->
<xsl:variable name="size">
```

```
    <xsl:choose>
        <xsl:when test="$x &gt; 3">большой</xsl:when>
        <xsl:otherwise>маленький</xsl:otherwise>
    </xsl:choose>
</xsl:variable>

<!-- XSLT 2.0 -->
<xsl:variable name="size" select="if ($x gt 3) then 'большой' else
'маленький' "/>
```

Полагаю, что большинство читателей предпочтет второе решение, возможное в XSLT 2.0.

Следует сделать важное замечание об условных выражениях в XPath: ветвь else обязательна. Программисты на языке C могут провести аналогию с тернарным выражением `a ? b : c`. Часто в случае, когда разумной альтернативы нет, используют пустую последовательность `()`.

Условные выражения удобны, когда нужно подставить значения по умолчанию, а схемы, в которой они были бы определены, не существует.

```
(: Значение по умолчанию для необязательного атрибута :)
if (@optional) then @optional else 'some-default'
(: Значение по умолчанию для необязательного элемента :)
if (optional) then optional else 'some-default'
```

Также они оказываются полезны, чтобы избежать неопределенных или нежелательных результатов при вычислении выражений. В следующем примере у нас, вероятно, есть причина предпочесть в качестве результата нуль, а не бесконечность `number('Infinity')`:

```
if ($divisor ne 0) then $dividend div $divisor else 0
```

Условия могут и более сложными. В следующем коде производится перекодировка списка из нескольких значений:

```
if (size eq 'XXL') then 50
else if (size eq 'XL') then 45
else if (size eq 'L') then 40
else if (size eq 'M') then 34
else if (size eq 'S') then 32
else if (size eq 'XS') then 29
else -1
```

Однако в данном случае предпочтительнее решение, основанное на последовательностях, особенно если заменить литеральные строки переменными, которые читаются из внешнего XML-файла:

```
(50,45,40,34,32,29,-1)[(index-of(('XXL','XL','L','M',S','XS')),
size),7)[1]]
```

Здесь предполагается, что у контекстного узла есть единственный дочерний элемент с именем `size`, в противном случае выражение окажется некорректным (хотя это можно исправить, написав `size[1]`). Мы также полагаемся на то, что функция `index-of` возвращает пустую последовательность, если элемент не найден. Именно для этого мы добавили заведомо отсутствующий элемент 7, реализовав тем самым поведение ветви `else`.

1.5. Исключение рекурсии с помощью выражений for

Задача

Требуется породить выходную последовательность из входной, причем каждый элемент выходной последовательности может быть произвольно сложной функцией от входных данных и размеры последовательностей не обязательно совпадают.

Решение

XPath 1.0

К версии 1.0 неприменимо. Необходимо использовать рекурсивный XSLT-шаблон.

XPath 2.0

Воспользуемся выражением for. В следующих примерах показано, как с помощью этого выражения можно отображать одну последовательность на другую, причем их размеры могут отличаться.

Агрегирование

```
(: Сумма квадратов :)
sum(for $x in $numbers return $x * $x)

(: Усреднение квадратов :)
avg(for $x in $numbers return $x * $x)
```

Отображение

```
(: Отобразить последовательность слов во всех абзацах на последователь-
ность длин слов. :)
for $x in //para/tokenize(.,' ') return string-length($x)

(: Отобразить последовательность слов в абзаце на последовательность их
длин, но только для слов, содержащих более 3 букв. :)
for $x in //para/tokenize(.,' ') return if (string-length($x) gt 3) then
string-length($x) else ()

(: То же самое, но с условием на входную последовательность. :)
```

```
for $x in //para/tokenize(.,' ')[string-length(.) gt 3] return string-
length($x)
```

Порождение

```
(: Породить последовательность квадратов первых 100 целых чисел. :)
for $i in 1 to 100 return $i * $i

(: Породить последовательность квадратов в обратном порядке. :)
for $i in 1 to 10 return (10 - $i) * (10 - $i)
```

Расширение

```
(: Отобразить последовательность абзацев на последовательность,
в которой каждый абзац повторяется дважды. :)
for $i in //para return ($x, $x)

(: Продублировать слова. :)
for $x in //para/tokenize(.,' ') return ($x, $x)

(: Сопоставить каждому слову это же слово, за которым следует его длина. :)
for $x in //para/tokenize(.,' ') return ($x, string-length($x))
```

Соединение

```
(: Для каждого клиента вывести идентификатор и общую сумму по всем его
заказам. :)
for $cust in doc('customer.xml')/*/customer
    return
       ($cust/id/text(),
        sum(for $ord in (doc('orders.xml')/*/order[custID eq $cust/id]
                 return ($ord/total)) )
```

Обсуждение

В рецепте 1.4 я уже отмечал, что добавление конструкций для управления потоком выполнения программы в язык выражений может поначалу показаться странным, а то и ошибочным решением. Но по мере овладения всей мощью XPath 2.0 эти сомнения отпадут. Особенно это относится к выражению `for`.

Достоинства выражения `for` становятся очевидны, когда вы понимаете, как с его помощью можно упростить многие рекурсивные решения, характерные для XSLT 1.0. Рассмотрим задачу вычисления сумм в XSLT 1.0. Если вам нужно вычислить простую сумму значений, то проблемы не возникает, так как в XSLT 1.0 есть встроенная функция `sum`. Но уже для вычисления суммы квадратов приходится писать более громоздкий и не сразу понятный рекурсивный шаблон. Вообще-то, значительная часть рецептов в первом издании этой книги как раз и была посвящена готовым решениям таких упражнений на применение рекурсии. При наличии XPath 2.0 сумма квадратов вычисляется проще пареной репы: `sum(for $x in $numbers return $x * $x)`, где `$numbers` – исходная

последовательность чисел. Подумайте, сколько деревьев я мог бы сохранить, если бы такая возможность существовала в XPath 1.0!

Однако этим мощь выражения `for` не исчерпывается. Итерировать можно не только по одной переменной. С помощью нескольких переменных можно организовывать вложенные циклы для порождения последовательностей из взаимосвязанных узлов сложного документа.

```
(: Возвращает последовательность, содержащую идентификаторы абзацев
и идентификаторы тех абзацев, на которые имеются ссылки. :)
for $s in /*/section,
     $p in $s/para,
     $r in $p/ref
           return ($p/@id, $r)
```

Вы должны понимать, что семантически это выражение эквивалентно следующему, хотя и более компактно:

```
for $s in /*/section
           return $p in $s/para
                  return for $r in $p/ref
                  return ($p/@id, $r)
```

Заметим также, что необязательно использовать вложенные выражения `for`, когда порождаемую последовательность можно более элегантно представить с помощью традиционного путевого выражения.

```
(: Это нагромождение for — всего лишь многословный эквивалент выражения
/*/section/para/ref. :)
for $s in /*/section,
     $p in $s/para,
     $r in $p/ref return $r
```

Иногда возникает необходимость узнать позицию каждого обрабатываемого элемента в последовательности. Воспользоваться для этого функцией `position()`, как в конструкции `for-each`, не получится, поскольку при вычислении выражения `for` в XPath позиция контекстного узла не изменяется. Но того же результата можно достичь следующим образом:

```
for $pos in 1 to count($sequence),
     $item in $sequence[$pos]
                return $item, $pos
```

1.6. Упрощение сложной логики с помощью кванторов

Задача

Требуется проверить выполнение некоторого условия для некоторых или всех элементов последовательности.

Решение

XPath 1.0

Если условие сводится к проверке на равенство, то достаточно семантики операторов = и != в XPath 1.0 и 2.0.

```
(: Истинно, если есть ссылка хотя бы на один раздел. :)
//section/@id = //ref/@idref

(: Истинно, если на каждый элемент section ссылается хотя бы один
элемент ref. :)
//count(//section) = count(//section[@id = //ref/@idref])
```

XPath 2.0

В XPath 2.0 того же результата можно достичь с помощью выражений `some` и `any`.

```
(: Истинно, если есть ссылка хотя бы на один раздел. :)
some $id in //section/@id satisfies $id = //ref/@idref

(: Истинно, если на каждый элемент section ссылается хотя бы один эле-
мент ref. :)
every $id in //section/@id satisfies $id = //ref/@idref
```

Однако без особых усилий можно пойти гораздо дальше.

```
(: Существует раздел, который ссылается на любой раздел, кроме себя
самого. :)
some $s in //section satisfies
    every $id in //section[@id ne $s/@id]/@id satisfies
            $id = $s/ref/@idref

(: $sequence2 является подпоследовательностью $sequence1. :)
count($sequence2) <= count($sequence1) and
    every $pos in 1 to count($sequence1),
            $item in $sequence1[$pos],
            $item in $sequence2[$pos] satisfies $item1 = $item2
```

Если удалить из этого выражения проверку `count`, то оно будет означать, что по крайней мере первые `count($sequence1)` элементов последовательности `$sequence2` совпадают с соответствующими элементам `$sequence1`.

Обсуждение

Семантика операторов =, !=, <, >, <=, >= в XPath 1.0 и 2.0 иногда приводит новичков в недоумение, если один из операндов является последовательностью или набором узлов в смысле XPath 1.0. Дело в том, что эти операторы возвращают true, если хотя бы одна пара значений с каждой стороны удовлетворяет

условию. В XPath 1.0 это бывает полезно, как мы раньше видели, но чаще заставляет чесать в затылке и мечтать о возвращении в пятый класс, где математика была ясной и понятной. Например, всякий сказал бы, что выражение `$x = $x` всегда истинно, но, если `$x` – пустая последовательность, то это не так! А все потому, что, раз последовательность пуста, то нет ни одной пары равных значений.

1.7. Операции над множествами

Задача

Требуется обработать последовательность, как если бы она была математическим множеством.

Решение

XPath 1.0

Операция объединения (|) наборов узлов поддерживается и в XPath 1.0, но, чтобы вычислить пересечение или разность, придется проявить изобретательность.

```
(: объединение :)
$set1 | $set2

(: пересечение :)
$set1[count(. | $set2) = count($set2)]

(: разность :)
$set1[count(. | $set2) != count($set2)]
```

XPath 2.0

Оператор | остался и в версии XPath 2.0, но добавился еще синоним для него `union`. Также появились операторы `intersect` и `except`, вычисляющие пересечение и разность соответственно.

```
$set1 union $set2

(: пересечение :)
$set1 intersect $set2

(: разность :)
$set1 except $set2
```

Обсуждение

В XPath 2.0 наборы узлов заменены последовательностями. Они, в отличие от наборов узлов, упорядочены и могут содержать дубликаты. Однако в случае

применения к ним теоретико-множественных операций дубликаты устраняются, а порядок игнорируется, так что последовательности ведут себя как обычные множества. Результат операции никогда не содержит дубликатов, даже если в исходных последовательностях они были.

Оператор `except` идиоматически используется в XPath 2.0 для отбора всех атрибутов, кроме входящих в заданное множество:

```
(: Все атрибуты, кроме @a :)
@a* except @a

(: Все атрибуты, кроме @a и @b :)
@a* except @a, @b
```

В версии 1.0 решение будет более громоздким:

```
@*[local-name(.) != 'a' and local-name(.) != 'b']
```

Интересно, что в XPath теоретико-множественные операции разрешены только над последовательностями узлов, а к атомарным значениям неприменимы. Связано это с тем, что операции выполняются над идентификаторами узлов, а не над значениями. Имитировать операции над значениями в XPath 2.0 можно с помощью показанных ниже выражений. В XPath 1.0 для этого потребуется рекурсия (см. главу 8).

```
(: объединение :)
distinct-values( ($items1, $items2) )

(: пересечение :)
distinct-values( ($items1[. = $items2] )

(: разность :)
distinct-values( ($items1[not(. = $items2)] )
```

См. также

В рецептах 9.1 и 9.2 приведены дополнительные примеры операций над множествами.

1.8. Сравнение узлов

Задача

Требуется идентифицировать узлы или сравнить их на основе позиции в документе.

Решение

XPath 1.0

В следующих примерах предполагается, что `$x` и `$y` содержат по одному узлу из одного и того же документа. Напомним также, что *порядок документа* – это тот

порядок, в котором узлы следуют в документе.

```
(: Проверить, представляют ли переменные $x и $y один и тот  же узел. :)
generate-id($x) = generate-id($y)

(: Можно также воспользоваться тем фактом, что оператор | удаляет
дубликаты. :)
count($x|$y) = 1

(: Проверить, что $x предшествует $y в порядке документа. Отметим,
что этот способ не работает, если $x или $y - атрибут. :)
count(($x/preceding::node()) < count(($y/preceding::node()) or
$x = $y/ancestor::node()

(: Проверить, что $x следует за $y в порядке документа. Отметим,
что этот способ не работает, если $x или $y — атрибут. :)
count(($x/following::node()) < count(($y/following::node()) or
$x = $y/ancestor::node()
```

XPath 2.0

```
(: Проверить, представляют ли переменные $x и $y один и тот  же узел. :)
$x is $y

(: Проверить, что $x предшествует $y в порядке документа. :)
$x << $y

(: Проверить, что $x следует за $y в порядке документа. :)
$x >> $y
```

Обсуждение

Появившиеся в XPath 2.0 операторы сравнения узлов работают более эффективно и проще для понимания, чем конструкции из XPath 1.0. Однако при работе с XPath 2.0 ситуации, когда в этих операторах возникает необходимость, встречаются не так уж часто. Обычно, когда вам кажется, что нужно бы воспользоваться оператором << или >>, задачу проще решить с помощью предложения `xsl::for-each-group`. Примеры см. в рецепте 6.2.

1.9. Как ужиться с расширенной системой типов в XPath 2.0

Задача

Более строгая типизации в XPath 2.0 заставляет вас проклинать W3C и с тоской вспоминать о Perl.

Решение

Большая часть несовместимостей между версиями XPath/XSLT 1.0 и 2.0 проистекает из-за ошибок типизации, причем это даже не зависит от наличия или отсутствия схемы. Многих проблем, которые связаны с XPath и возникают при переносе кода в версию 2.0, можно избежать, если запускать программу в режиме совместимости с версией 1.0.

```
<xsl:stylesheet version="1.0">

    <!-- ... -->

</xsl:stylesheet>
```

Мне думается, что со временем вы захотите отказаться от режима совместимости. В XPath 2.0 есть несколько способов выполнить преобразование типов. Во-первых, можно напрямую обратиться к функциям преобразования.

```
(: Преобразовать первый дочерний элемент X контекстного узла в число. :)
number(X[1]) + 17

(: Преобразовать число, хранящееся в переменной $n, в строку. :)
concat("id-", string($n))
```

В XPath 2.0 есть также конструкторы типов, так что управлять интерпретацией строки можно явно.

```
(: Сконструировать дату из строки. :)
xs:date("2005-06-01")

(: Сконструировать число с двойной точностью из строки. :)
xs:double("1.1e18") + xs:double("23000")
```

Наконец, в XPath есть операторы `castable as`, `cast as` и `treat as`. Обычно бывает достаточно двух первых.

```
if ($x castable as xs:date) then $x cast as xs:date else
xs:date("1970-01-01")
```

Оператор `treat as` не выполняет преобразования типов, а служит утверждением, которое процессор XPath воспринимает как обещание подавать во время выполнения значение указанного типа. Если обещание будет нарушено, возникнет ошибка. Оператор `treat as` был введен для того, чтобы разработчики XPath могли реализовать статическую (во время компиляции) проверку типов в дополнение к динамической проверке, которая позволила бы программистам избирательно отключать статический контроль. В реализациях XSLT 2.0 статический контроль типов, скорее всего, будет редкостью, так что пока можно спокойно игнорировать оператор `treat as`. Необходимость

в нем гораздо вероятнее в более продвинутых процессорах языка XQuery, где статический контроль позволяет выполнять различные оптимизации.

Обсуждение

Запуск процессора XSLT 2.0 в режиме совместимости с версией 1.0 не означает, что новые средства 2.0 окажутся недоступными. В этом режиме просто активируются некоторые правила преобразования, существовавшие в XPath 1.0.

- Допускается использовать нечисловые типы в контексте, где ожидается число. При этом автоматически выполняется преобразование, заключающееся в приведении к атомарному значению с последующим вызовом функции `number()`.

```
(: В режиме совместимости вычисление следующего выражения дает 18.1,
а в версии 2.0 приведет к ошибке. :)
"1.1" + "17"
```

- Допускается использование не-строковых типов в контексте, где ожидается строка. При этом автоматически выполняется преобразование, заключающееся в приведении к атомарному значению с последующим вызовом функции `string()`.

```
(: В режиме совместимости вычисление следующего выражения дает 2,
а в версии 2.0 приведет к ошибке. :)
string-length(1 + 2 + 3 + 4 + 5)
```

- Из последовательностей длины 2 и более автоматически удаляются все элементы, кроме первого, в контексте, где ожидается единственное значение. Так часто бывает при передаче функции результата вычисления путевого выражения.

```
<poem>
    <line>There once was a programmer from Nantucket.</line>
    <line>Who liked his bits in a bucket.</line>
    <line>He said with a grin</line>
    <line>and drops of coffee on his chin,</line>
    <line>"if XSLT had a left-shift, I would love it!"</line>
</poem>

(: В режиме совместимости оба выражения дают 43, а в версии 2.0 первое
приводит к ошибке. :)
string-length(/poem/line)
string-length(/poem/line[1])
```

1.10. Как воспользоваться расширенной системой типов в XPath 2.0

Задача

Вы скрупулезно применяете схемы при обработке XML-документов и хотели бы пожать плоды своих трудов.

Решение

Если вы проверяете документы на соответствие схеме, то получающиеся узлы аннотируются информацией о типе. Затем в XPath 2.0 (а равно при сопоставлении с шаблонами в XSLT 2.0) эти типы можно опросить.

```
(: Проверить, что все элементы invoiceDate действительно являются
датами. :)
if (order/invoiceDate instance of element(*, xs:date)) then "накладная
корректна" else "накладная некорректна"
```

Оператор instance of полезен *только* при наличии схемы. И его семантика отличается от оператора `castable as`. Например, выражение `10 castable as xs:positiveInteger` всегда дает true, тогда как выражение `10 instance of xs:positiveInteger` ложно, поскольку литеральные числовые значения аннотируются типом `xs:decimal`.

Но проверка на соответствие схеме полезна не только потому, что вы получаете возможность опрашивать типы. Всегда безопаснее и удобнее заранее знать, что при обработке документа не возникнет никаких сюрпризов из-за ошибок типизации. В результате можно писать более компактные шаблоны.

```
(: Без проверки пришлось бы писать такой код :)
for $order in Order return xs:date($order/invoicedate) - xs:date($order/
createdate)

(: Если известно, что все даты проверены, то можно обойтись
без конструктора xs:date :)
for $order in Order return $order/invoicedate - $order/createdate
```

Обсуждение

Лично я предпочитаю использовать XML-схемы как спецификации документов, а не как инструмент проверки. Поэтому я стараюсь писать XSLT-преобразования, устойчивые к ошибкам типизации, и при необходимости применяю явные преобразования. Так составленные таблицы стилей будут работать вне зависимости от того, была проведена предварительная проверка или нет.

Начав писать таблицы стилей, зависящие от проверки, вы привязываете себя только к таким реализациям, которые умеют проверять соответствие схеме. С другой стороны, если в компании принят стандарт, в соответствии с которым все XML-документы должны проверяться перед обработкой, то вы можете упростить свои XSLT-программы, так как есть уверенность, что в определенных контекстах будут присутствовать только данные заведомо известных типов.

Глава 2. Строки

> Я считаю, что все должны иметь оружие. У граждан должны быть базуки и реактивные установки. Я считаю, что каждый гражданин имеет право на выбор оружия. Но я также считаю, что только у меня могут быть боеприпасы. Поскольку, честно говоря, я бы не доверил всей этой деревенщине ничего более опасного, чем шнурки от ботинок[1].
>
> *Скотт Адамс*

2.0. Введение

В деле манипуляции строками XSLT 1.0, конечно же, не хватает тяжелой артиллерии Perl. Язык XSLT оптимизирован для обработки XML-разметки, а не строк. Но, поскольку XML – это просто структурированный текстовый формат, то строки с неизбежностью появляются во всех задачах преобразования, кроме самых простых. К сожалению, в XSLT 1.0 есть всего девять стандартных функций для обработки строк. Напротив, в Java их около двух десятков, а в Perl – бесспорном короле современных языков для обработки текстов – пара десятков плюс чрезвычайно богатый механизм регулярных выражений.

Но с выходом в свет реализаций XSLT 2.0 программисты XSLT могут перестать так уж сильно завидовать Perl-овикам. В XPath 2.0 есть 20 функций, относящихся к работе со строками, в том числе и поддержка регулярных выражений. Помимо того, добавлены средства для анализа неструктурированного текста с помощью регулярных выражений с целью преобразования его в правильный XML-документ.

У программистов на XSLT 1.0, нуждающихся в нетривиальной обработке строк, есть два варианта действий. Во-первых, можно вызывать внешние функции, написанные на Java или ином языке, который поддерживается имеющимся XSLT-процессором. Это очень удобно, если только не стоит вопрос о переносимости. Во-вторых, можно реализовать дополнительные средства обработки строк прямо на XSLT. В этой главе мы рассмотрим, что можно делать со строками в XSLT 1.0 и как те же задачи решаются в XSLT 2.0.

Для реализации дополнительных функций работы со строками в XSLT 1.0 имеющиеся функции сочетаются с рекурсией, которая является неотъемлемой частью всех сколько-нибудь содержательных применений XSLT. На самом деле, рекурсия – это такой важный компонент XSLT, что некоторые из рецептов в этой главе имеет смысл изучить, даже если вы не собираетесь реализовывать обработку строк непосредственно на этом языке.

---

[1] Слово *string* имеет несколько значений, в частности: *строка* и *шнурок*. (*Прим. перев.*)

В этой книге мы часто ссылаемся на великолепный проект EXSLT.org, созданный по инициативе сообщества пользователей, желающих стандартизировать расширения XSLT. Рекомендую заглянуть на сайт проекта по адресу *http://www.exslt.org*.

Занимаясь задачей, для решения которой требуется больше пары строк кода, я всегда применяю появившуюся в XSLT 2.0 возможность писать полноценные функции на языке XPath. В XSLT 1.0 для этой цели используются именованные шаблоны, вызываемые с помощью `xsl:call-template`.

2.1. Завершается ли данная строка указанной подстрокой?

Задача

Требуется проверить, завершается ли некоторая строка указанной подстрокой.

Решение

XSLT 1.0

```
substring($value, (string-length($value) - string-length($substr)) + 1)
= $substr
```

XSLT 2.0

```
ends-with($value, $substr)
```

Обсуждение

В XSLT 1.0 есть встроенная функция `starts-with()`, но нет функции `ends-with()`. В версии 2.0 это исправлено. Однако, как видно из примера выше, функцию `ends-with()` в XPath 1.0 можно реализовать в терминах функций `substring()` и `string-length()`. Мы просто извлекаем последние `string-length($substr)` символов из проверяемой строки и сравниваем их с подстрокой.

Программисты, привыкшие к тому, что нумерация символов в строке начинается с 0, должны помнить, что в XSLT номер первой позиции равен 1.

2.2. Нахождение позиции начала подстроки

Задача

Требуется найти индекс позиции, с которой начинается подстрока, а не текст до или после этой подстроки.

Решение

XSLT 1.0

```
<xsl:template name="str:index-of">
   <xsl:param name="input"/>
   <xsl:param name="substr"/>
<xsl:choose>
   <xsl:when test="contains($input, $substr)">
   <xsl:value-of select="string-length(substring-
       before($input, $substr))+1"/>
   </xsl:when>
   <xsl:otherwise>0</xsl:otherwise>
</xsl:choose>
</xsl:template>
```

XSLT 2.0

```
<xsl:function name="ckbk:string-index-of">
  <xsl:param name="input"/>
  <xsl:param name="substr"/>
  <xsl:sequence select="if (contains($input, $substr))
                        then string-length(substring-before($input, $substr))+1
                         else 0"/>
</xsl:function>
```

Обсуждение

Позиция начала подстроки – это длина предшествующей ей строки плюс 1. Если вы точно знаете, что исходная строка содержит данную подстроку, то можете просто вычислить выражение `string-length(substring-before($input, $substr))+1`. Но в общем случае нужно обрабатывать также случай, когда подстрока не входит в строку. Мы в этой ситуации возвращаем 0, но можно выбрать и какой-нибудь другой индикатор, например, -1 или `NaN`.

2.3. Удаление заданных символов из строки

Задача

Требуется убрать из строки некоторые символы (например, пробельные).

Решение

XSLT 1.0

Воспользуйтесь функцией `translate()` с пустой строкой замены. Например, следующий код удаляет из строки все пробельные символы:

```
translate($input, " &#x9;&#xa;&#xd;", "")
```

XSLT 2.0

Применение `translate` оправдано и в XSLT 2.0, так как обычно эта функция работает быстрее всего. Но некоторые задачи такого рода более естественно решаются с помощью регулярных выражений и новой функции `replace()`:

```
(: \s соответствует любому пробельному символу :)
replace($input, "\s", "")
```

Обсуждение

`translate()` – довольно гибкая функция, которая часто применяется для компенсации отсутствующих в XSLT 1.0 средств работы со строками. Здесь мы воспользовались тем фактом, что `translate()` не копирует те символы входной строки, которые есть в строке `from`, но отсутствуют в строке `to`.

Функцию translate() можно использовать также для удаления из строки всех символов, кроме заданных. Например, следующий код удаляет из исходной строки все символы, кроме цифр:

```
translate($string,
          translate($string, '0123456789', ''), '')
```

Внутренний вызов `translate()` удаляет все интересующие нас символы (в данном случае цифры), чтобы получить параметр `from` для внешнего вызова, который удалит из исходной строки все символы, кроме цифр.

Иногда нужно удалять не все пробелы, а только находящиеся в начале и в конце строки, а также оставлять из нескольких подряд идущих пробелов в середине только один. В XPath для этой цели есть встроенная функция `normalize-space()`. А если нужно решить ту же задачу для символа, отличного от пробела, то можно воспользоваться таким кодом (в данном случае мы нормализуем вхождения символа `C`):

```
translate(normalize-space(translate($input,"C "," C")),"C "," C")
```

Однако это преобразование будет работать неправильно, если входная строка содержит другие символы пропуска, то есть знаки табуляции (#x9), конца строки (#xA) и перевода каретки (#xD). Причина в том, что это выражение переставляет местами пробел и нормализуемый символ, затем нормализует пробелы и выполняет обратную перестановку. Если после первого преобразования остается символ пропуска, отличный от пробела, то он также нормализуется, хотя вы, возможно, этого и не хотели. Впрочем, нормализация чего-либо, кроме пробелов, встречается редко. Ниже показано, как можно удалить лишние символы -:

```
<xsl:template match="/">
  <xsl:variable name="input" select=" '-this -is- the way we normalize
non-whitespace-' "/>
  <xsl:value-of select="translate(normalize-space(
                                translate($input,'- "," -')),'- "," -')"/>
</xsl:template>
```

XSLT 2.0

Более универсальный способ удаления ненужных символов дает встроенная в XSLT 2.0 функция `replace()`, основанная на аппарате регулярных выражений. Ниже мы с помощью `replace()` нормализуем отличный от пробела символ без особых случаев, возникающих в решении для XSLT 1.0:

```
<xsl:template match="/">
    <xsl:variable name="input"
        select=" '-this -is- the way we normalize non-whitespace-' "/>
    <xsl:value-of selec"="replace(replace($input,'-+','-'),'^-+|-+$','')"/>
</xsl:template>
```

Здесь функция `replace()` вызывается дважды. Внутренний вызов заменяет все соседние вхождения символа одним, а внешний удаляет начальные и конечные символы.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

В этой главе мы познакомимся с одним из инструментов, которыми опытные программисты обожают пользоваться для манипуляции строками: регулярными выражениями (их еще любовно называет просто regex). Какого бы разработчика на XSLT я ни спрашивал, добавление регулярных выражений в XSLT стояло в верхней позиции его списка из 10 наиболее желательных усовершенствований. Эта врезка предназначена для тех программистов, которые еще не имели удовольствия работать с регулярными выражениями или побаиваются их. Справочник не исчерпывающий, но для начала его хватит.

Регулярное выражение – это строка, содержащая образец для сопоставления с другой строкой. Простейший образец – литеральная строка, то есть строка «foo» тоже считается регулярным выражением. Оно может быть сопоставлено, к примеру, строке «foobar», начиная с первого символа. Но истинная мощь регулярных выражений проявляется, когда вы включаете специальные метасимволы, распознаваемые языком.

Ниже перечислены наиболее распространенные метасимволы.

- Точка (.) сопоставляется с любым одиночным символом.
- Класс символов (`[aeiou]`, `[a-z]` или `[A-Z]`) сопоставляется с одиночным символом, принадлежащим одному или нескольким диапазонам.
- Для некоторых классов символов имеются специальные сокращения. Например, `\s` – сокращенная запись класса символов пропуска, к которым относятся пробел, а также знаки табуляции, новой строки и возврата каретки, а `\d` – сокращение для класса `[0-9]`. Часто для сокращения, включающего символ обратной косой черты и строчную букву, имеется парное сокращение с заглавной буквой; оно инвертирует сопоставление. Так, например, `\S` сопоставляется с любым символом, отличным от пропуска, а `\D` – со всеми символами, кроме цифр. Впрочем, это не универсальное правило. К примеру, `\n` сопоставляется с символом новой строки, но `\N` не означает сопоставления с любым символом, кроме новой строки (то же самое относится к обозначениям `\t` – табуляция и `\r` – возврат каретки).

- Можно инвертировать класс символов, поставив в начале знак ^. Например, `[^aeiou]` сопоставляется с любым символом, кроме строчных гласных букв. Это относится и к диапазонам; `[^0-9]` – то же самое, что `\D`.
- Литералы и метасимволы часто встречаются в сочетании. Например, `d[aeiou]g` сопоставляется со словами «`dag`», «`deg`», «`dig`», «`dog`» и «`dug`», а также с любой строкой, содержащий одну из вышеуказанных в качестве подстроки.
- Не менее важны метасимволы повторения, с помощью которых производится сопоставление с предшествующим им регулярным выражением, которое встречается несколько раз.
- Метасимвол * означает, что предыдущее выражение может встречаться 0 или более раз. Следовательно, `be*` сопоставляется со строками «b», «`be`», «`bee`», «`beee`» и так далее. Выражение `(10)*` сопоставляется со строками «`10`», «`1010`», «`101010`» и так далее. Здесь скобки применяются для группирования. Если скобки удалить, останется выражение `10*`, так что символ повтора будет применяться только к цифре `0`.
- Метасимвол + означает, что предыдущее выражение может встречаться 1 или более раз. Следовательно, `be+` сопоставляется со строками «`be`», «`bee`», «`beee`» и так далее, но не со строкой «`b`».
- Метасимвол ? означает, что предыдущее выражение может встречаться 0 или 1 раз. Следовательно, `be?` сопоставляется только со строками «`b`» и «`be`».
- Очень часто бывает необходимо точно указать, в каком месте строки может происходить сопоставление с регулярным выражением. В частности, нередко требуется, чтобы сопоставление происходило только в начале (^) или в конце ($) строки, а иногда нужно, чтобы строка считалась подходящей, если она начинается с образца и им же заканчивается. Например, выражение `^be+` сопоставляется со строкой «`bee keeper`», но не со строкой «`has been`». Выражение `be+$` сопоставляется со строкой «`to be or not to be`», но не со строкой «`be he alive or be he dead`». И, наконец, выражение `^be+$` сопоставляется со строками «`be`» и «`bee`», но не со строками «`been`» или «`Abe`».
- Представленные до сих пор конструкции позволяют решить большинство задач на сопоставление с образцом, с которыми вам придется столкнуться. Но есть еще и *контекстно-зависимые* сопоставления, для которых простых регулярных выражений недостаточно. Предположим, что нужно найти числа, начинающиеся и заканчивающиеся одной и той же цифрой (11, 909, 3233 и т.д.). Регулярные выражения в чистом виде с такой задачей не справятся, но в большинстве языков, в том числе и в XPath 2.0, реализованы расширения, позволяющие решить ее.

- Для применения описываемого механизма необходимо сделать две вещи. Во-первых, ту часть образца, на которую вы хотите в дальнейшем сослаться, нужно специально пометить как *запоминаемую группу*, для чего служат круглые скобки. А, во-вторых, на эту группу следует сослаться по номеру. Например, выражение `(\d)\d*\1` сопоставляется с любым числом, которое начинается и заканчивается одной и той же цифрой. В данном случае группа состоит из одной цифры (`\d`), а ссылка на нее имеет вид `\1`, то есть «первая сопоставленная группа». Вы, наверное, уже догадались, что групп может быть несколько, например, `(\d)(\d)\1\2`. Такое выражение сопоставляется с числами вида «`1212`» и «`9999`», но не с «`1213`» или «`1221`». Обратные ссылки `\1`, `\2` и т.д. применяются в функции `matches()`, включенной в XPath 2.0. Похожая нотация, в которой вместо знака \ используется $, зарезервирована для случаев, когда ссылка встречается вне регулярного выражения. Это характерно для функции `replace()`, в которой нужно сослаться на группы, описанные в регулярном выражении поиска, из регулярного выражения замены. Например, `replace($someText, '(\d)\d*)', '$1')` заменяет первую последовательность из одной или более цифр в строке `$someText` первой цифрой этой последовательности. Этот механизм доступен также из команды `xsl:analyze-string`. Подробнее мы рассмотрим его в рецептах 2.6 и 2.10.

Если вы захотите более глубоко познакомиться с регулярными выражениями, обратите внимание на книгу Jeffery E.F. Friedl *Mastering Regular Expressions, Second Edition* (O'Reilly, 1999)[1]. Если вас интересуют особенности регулярных выражений в XSLT 2.0, познакомьтесь с книгой Michael Kay *XPath 2.0* (Wrox, 2004) или с рекомендациями консорциума W3C по адресу http://www.w3.org/TR/xquery-operators#string.match и http://www.w3.org/TR/xmlschema-2#regexs.

2.4. Поиск подстроки с конца строки

Задача

В XSLT нет функций для поиска в строке, начиная с конца.

Решение

XSLT 1.0

С помощью рекурсии можно эмулировать поиск последнего вхождения подстроки `substr`. Эта техника позволяет написать шаблоны `substring-before-last` (выделение строки, предшествующей последнему вхождению) и substring-after-last (выделение строки, следующей за последним вхождению):

[1] Дж. Фридл «Регулярные выражения». Издательство «Питер», 2003.

```
<xsl:template name="str:substring-before-last">
  <xsl:param name="input"/>
  <xsl:param name="substr"/>

  <xsl:if test="$substr and contains($input, $substr)">
    <xsl:variable name="temp" select="substring-after($input, $substr)" />
    <xsl:value-of select="substring-before($input, $substr)" />
    <xsl:if test="contains($temp, $substr)">
      <xsl:value-of select="$substr" />
      <xsl:call-template name="str:substring-before-last">
        <xsl:with-param name="input" select="$temp" />
        <xsl:with-param name="substr" select="$substr" />
      </xsl:call-template>
    </xsl:if>
  </xsl:if>

</xsl:template>

<xsl:template name="str:substring-after-last">
  <xsl:param name="input"/>
  <xsl:param name="substr"/>

  <!-- Выделить строку, следующую за первым вхождением -->
  <xsl:variable name="temp" select="substring-after($input,$substr)"/>

  <xsl:choose>
    <xsl:when test="$substr and contains($temp,$substr)">
        <xsl:call-template name="str:substring-after-last">
             <xsl:with-param name="input" select="$temp"/>
             <xsl:with-param name="substr" select="$substr"/>
        </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
        <xsl:value-of select="$temp"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

XSLT 2.0

В XSLT 2.0 нет вариантов функций `substring-before/after`, которые позволяли бы искать от конца строки, но добиться желаемого результата позволяет функция `tokenize()`, основанная на применении регулярных выражений:

```
<xsl:function name="ckbk:substring-before-last">
    <xsl:param name="input" as "xs:string"/>
```

```
    <xsl:param name="substr" as "xs:string"/>
    <xsl:sequence
        select="if ($substr)
                then
                   if (contains($input, $substr)) then
                   string-join(tokenize($input, $substr)
                     [position() ne last()],$substr)
                    else ""
                else $input"/>
</xsl:function>

<xsl:function name="ckbk:substring-after-last">
    <xsl:param name="input" as "xs:string"/>
    <xsl:param name="substr" as "xs:string"/>
    <xsl:sequence
    select="if ($substr)
            then
               if (contains($input, $substr))
               then tokenize($input, $substr)[last()]
               else ''
            else $input"/>
</xsl:function>
```

В обеих функциях нужно проверять, не является ли строка `substr` пустой, поскольку функция `tokenize` не примет пустой образец для поиска. К сожалению, эти реализации работают не совсем так, как встроенные аналоги. Связано это с тем, что `tokenize` трактует свой второй аргумент как регулярное выражение, а не как литеральную строку. И это может стать источником неожиданностей. Можно исправить этот недостаток путем экранирования всех специальных символов, встречающихся в регулярном выражении. И включать или отключать такое поведение с помощью третьего булевского аргумента. Первоначальная версия с двумя аргументами и новая с тремя могут сосуществовать, так как XSLT допускает перегрузку функций (то есть функция полностью определяется своим именем и *арностью* (количеством аргументов).

```
<xsl:function name="ckbk:substring-before-last">
    <xsl:param name="input" as "xs:string"/>
    <xsl:param name="substr" as "xs:string"/>
    <xsl:param name="mask-regex" as "boolean"/>
    <xsl:variable name="matchstr"
              select="if ($mask-regex)
                         then replace($substr,'([.+?*^$])','\$1')
                         else $substr"/>

    <xsl:sequence select="ckbk:substring-before-last($input,$matchstr)"/>
```

```
</xsl:function>

<xsl:function name="ckbk:substring-after-last">
    <xsl:param name="input"/>
    <xsl:param name="substr"/>
    <xsl:param name="mask-regex"/>
    <xsl:variable name="matchstr"
             select="if ($mask-regex)
                        then replace($substr,'([.+?*^$])','\$1')
                        else $substr"/>

    <xsl:sequence select="ckbk:substring-after-last($input,$matchstr)"/>
</xsl:function>
```

Обсуждение

Обе функции поиска подстроки в XSLT (`substring-before` и `substring-after`) начинают поиск с начала строки. Но иногда нужно искать подстроку с конца строки. Проще всего решить эту задачу, рекурсивно применяя встроенные функции поиска, пока не будет найдено последнее вхождение подстроки.

В первой попытке написать эти шаблоны я столкнулся с неприятным эффектом, о котором вы должны помнить, когда работаете с рекурсивными шаблонами. Напомню, что выражение `contains($anything,'')` всегда возвращает `true`! Поэтому при рекурсивном вызове `substring-before-last` и `substring-after-last` я проверяю, что значение `$substr` не пусто. Без такой проверки мы попали бы в бесконечный цикл поиска пустой подстроки, а если реализация не поддерживает хвостовую рекурсию, то произошло бы переполнение стека.

Есть и другой алгоритм, который называется *разделяй и властвуй* или *деление пополам*. Его основная идея заключается в том, чтобы разбить строку на две половинки. Если искомая подстрока находится во второй половине, то первую можно не рассматривать и тем самым свести исходную задачу к другой, вдвое меньшей сложности. Этот процесс повторяется рекурсивно. Но нужно учесть еще случай, когда искомая строка частично находится в первой половине, а частично во второй. Ниже приведено решение для функции `substring-before-last`:

```
<xsl:template name="str:substring-before-last">

  <xsl:param name="input"/>
  <xsl:param name="substr"/>

  <xsl:variable name="mid" select="ceiling(string-length($input) div 2)"/>
  <xsl:variable name="temp1" select="substring($input,1, $mid)"/>
```

```
    <xsl:variable name="temp2" select="substring($input,$mid +1)"/>
    <xsl:choose>
     <xsl:when test="$temp2 and contains($temp2,$substr)">
        <!--искомая строка во второй половине, поэтому просто добавим
          первую половину и -->
        <!-- выполним рекурсивный вызов для второй -->
        <xsl:value-of select="$temp1"/>
        <xsl:call-template name="str:substring-before-last">
          <xsl:with-param name="input" select="$temp2"/>
          <xsl:with-param name="substr" select="$substr"/>
        </xsl:call-template>
     </xsl:when>
     <!--искомая строка на границе, задача решается простым вызовом
       substring-before-->
     <xsl:when test="contains(substring($input,
                                        $mid - string-length($substr) +1),
                                        $substr)">
        <xsl:value-of select="substring-before($input,$substr)"/>
     </xsl:when>
     <!--искомая строка в первой половине, поэтому вторую отбрасываем-->
     <xsl:when test="contains($temp1,$substr)">
        <xsl:call-template name="str:substring-before-last">
        <xsl:with-param name="input" select="$temp1"/>
        <xsl:with-param name="substr" select="$substr"/>
        </xsl:call-template>
      </xsl:when>
      <!-- Искомая строка не найдена, завершаемся -->
      <xsl:otherwise/>
    </xsl:choose>

</xsl:template>
```

Выясняется, что такой алгоритм деления пополам дает ощутимый выигрыш, только если просматриваемый текст достаточно велик (порядка 4000 символов и более). Можно написать шаблон-обертку, который будет выбирать подходящий алгоритм в зависимости от длины текста или переключаться с алгоритма деления пополам на более простой, если очередная часть оказывается достаточно короткой.

2.5. Повторение строки N раз

Задача

Требуется повторить строку N раз, где N – параметр. Например, нужно дополнить строку пробелами, чтобы выровнять ее по правому или левому краю.

Решение

XSLT 1.0

Задачу можно решить красиво, применив рекурсивный алгоритм, который удваивает строку, пока не будет достигнута нужная длина. Надо только аккуратно рассмотреть случай, когда значение `$count` нечетно.

```
<xsl:template name="dup">
    <xsl:param name="input"/>
    <xsl:param name="count" select="2"/>
    <xsl:choose>
         <xsl:when test="not($count) or not($input)"/>
         <xsl:when test="$count = 1">
              <xsl:value-of select="$input"/>
         </xsl:when>
         <xsl:otherwise>
              <!-- Если $count нечетно, добавить еще одну копию
              входной строки -->
              <xsl:if test="$count mod 2">
                   <xsl:value-of select="$input"/>
              </xsl:if>
              <!-- Рекурсивно применяем шаблон, предварительно удвоив
              входную строку и вдвое -->
              <!-- уменьшив счетчик -->
              <xsl:call-template name="dup">
                   <xsl:with-param name="input"
                        select="concat($input,$input)"/>
                   <xsl:with-param name="count"
                        select="floor($count div 2)"/>
              </xsl:call-template>
         </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

XSLT 2.0

В версии 2.0 дублирование легко реализуется с помощью выражения `for`. Мы перегружаем функцию `dup` с целью имитировать имеющий значение по умолчанию аргумент в реализации для версии XSLT 1.0.

```
<xsl:function name="ckbk:dup">
    <xsl:param name="input" as="xs:string"/>
    <xsl:sequence select="ckbk:dup($input,2)"/>
</xsl:function>

<xsl:function name="ckbk:dup">
  <xsl:param name="input" as="xs:string"/>
```

```
  <xsl:param name="count" as="xs:integer"/>
  <xsl:sequence select="string-join(for $i in 1 to $count return $input,'')"/>
</xsl:function>
```

Обсуждение

XPath 1.0

Самый очевидный способ продублировать строку `$count` раз – конкатенировать ее саму с собой `$count - 1` раз. Это можно сделать рекурсивно, как показано ниже, но такой код работает неэффективно для сколько-нибудь большого числа повторений, поэтому применять его не рекомендуется.

```
<xsl:template name="slow-dup">
    <xsl:param name="input"/>
    <xsl:param name="count" select="1"/>
    <xsl:param name="work" select="$input"/>
    <xsl:choose>
        <xsl:when test="not($count) or not($input)"/>
        <xsl:when test="$count=1">
            <xsl:value-of select="$work"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:call-template name="slow-dup">
                <xsl:with-param name="input" select="$input"/>
                <xsl:with-param name="count" select="$count - 1"/>
                <xsl:with-param name="work"
                    select="concat($work,$input"/>
            </xsl:call-template>
       </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Более разумный подход приведен в разделе «Решение». В нем число рекурсивных вызовов и конкатенаций уменьшено примерно до `log2($count)` за счет повторного удвоения входной строки и уменьшения счетчика вдвое до тех пор, пока он не станет равен 1. Реализация `slow-dup` громоздка еще и потому, что требует искусственного параметра `work`, в котором сохраняется исходная строка. Кроме того, вследствие рекурсии стек растет до `$count - 1` уровней и требуется `$count - 1` обращений к `concat`. А в реализации `dup` рост стека ограничен `floor(log2($count))` уровнями и требуется лишь `ceqling(log2($count))` вызовов `concat`.

Техника, применяемая в `slow-dup`, имеет одно достоинство – она позволяет дублировать не только строки, но и произвольные структуры. Достаточно заменить `xsl:value-of` на `xsl:copy-of`. Более быстрая версия `dup` лишена этого преимущества, так как копии передаются как параметры, а это обходится дорого.

Ниже приведено еще одно решение, основанное на идее функции `str:padding` из проекта EXSLT, но не совпадающее с ней дословно:

```
<xsl:template name="dup">
    <xsl:param name="input"/>
    <xsl:param name="count" select="1"/>
    <xsl:choose>
      <xsl:when test="not($count) or not($input)" />
      <xsl:otherwise>
         <xsl:variable name="string"
                       select="concat($input, $input, $input, $input,
                                      $input, $input, $input, $input,
                                      $input, $input)" />
         <xsl:choose>
            <xsl:when test="string-length($string) >=
                            $count * string-length($input)">
               <xsl:value-of select="substring($string, 1,
                                    $count * string-length($input))" />
            </xsl:when>
            <xsl:otherwise>
               <xsl:call-template name="dup">
                  <xsl:with-param name="input" select="$string" />
                  <xsl:with-param name="count" select="$count div 10" />
               </xsl:call-template>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

В этой реализации делается десять копий входной строки. Если это больше, чем необходимо, то результат усекается до нужной длины. В противном случае шаблон применяется рекурсивно. Это решение работает медленнее, так как часто производится больше конкатенаций, чем необходимо, и вызывается функция `substring()`, которая в некоторых реализациях XSLT не эффективна. Объяснение см. в рецепте 2.7. Зато оно обладает преимуществами при исполнении процессорами, которые не оптимизируют хвостовую рекурсию, поскольку число рекурсивных вызовов заметно меньше.

См. также

Так называемый *метод Пиза* также позволяет дублировать строку без рекурсии. Он обсуждается в документе *http://www.xml.org/xml/xslt_efficient_programming_techniques.pdf*. Суть его в том, чтобы использовать цикл `for-each` для любого доступного источника узлов (часто самой таблицы стилей). Хотя на практике этот

метод может показать высокую эффективность, я считаю его несовершенным из-за предположения о том, что узлов достаточно для выполнения требуемого числа итераций.

2.6. Обращение строки

Задача

Требуется изменить порядок символов в строке на противоположный.

Решение

XSLT 1.0

Приведенный ниже шаблон обращает строку `$input` неочевидным, но весьма эффективным способом.

```
<xsl:template name="reverse">
    <xsl:param name="input"/>
    <xsl:variable name="len" select="string-length($input)"/>
    <xsl:choose>
         <!-- Строки длиной меньше 2 обращаются тривиально -->
         <xsl:when test="$len &lt; 2">
            <xsl:value-of select="$input"/>
         </xsl:when>
         <!-- Строки длины 2 также обращаются тривиально -->
         <xsl:when test="$len = 2">
             <xsl:value-of select="substring($input,2,1)"/>
             <xsl:value-of select="substring($input,1,1)"/>
         </xsl:when>
         <xsl:otherwise>
                  <!-- Шаблон рекурсивно применяется сначала ко второй,
                  а потом к первой половине входной строки -->
                  <xsl:variable name="mid" select="floor($len div 2)"/>
                  <xsl:call-template name="reverse">
                       <xsl:with-param name="input"
                           select="substring($input,$mid+1,$mid+1)"/>
                  </xsl:call-template>
                  <xsl:call-template name="reverse">
                      <xsl:with-param name="input"
                          select="substring($input,1,$mid)"/>
                  </xsl:call-template>
           </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

XSLT 2.0

В версии 2.0 обращение строки выполняется тривиально.

```
<xsl:function name="ckbk:reverse">
    <xsl:param name="input" as="xs:string"/>
    <xsl:sequence select="codepoints-to-string(
                            reverse(string-to-codepoints($input)))"/>
   </xsl:function>
```

Обсуждение

XSLT 1.0

Приведенный алгоритм не самый очевидный, зато эффективный. Он успешно обращает даже очень длинные строки, на которые у других, более понятных, алгоритмов уходит слишком много времени или дело вообще кончается переполнением стека. Основная идея заключается в том, чтобы переставить две половины строки местами и продолжать рекурсивное применение алгоритма к каждой половине, пока не останутся строки длины 2 или менее, обращение которых тривиально. Следующий пример иллюстрирует работу алгоритма. На каждом шаге знаком + отмечено место расщепления и конкатенации строки.

1. reverse(«abcdef») (входная строка)
2. reverse(«def») + reverse(«abc»)
3. reverse(«ef») + «d» + reverse(«bc») + «a»
4. «f» + «e» + «d» + «c» + «b» + «a»
5. fedcba (результат)

Поучительно будет рассмотреть более понятные реализации обращения строки на языке XSLT. Это покажет вам, как следует и как не следует реализовывать рекурсивные решения в других контекстах.

Один из самых худших алгоритмов – тот, что в первый момент приходит в голову. Его идея состоит в том, чтобы переставить местами первый и последний символ, затем второй и предпоследний и продолжать так до тех пор, пока не дойдем до середины строки. Такое решение мог бы предложить программист на языке C, поскольку оно очень эффективно в языке, где можно получить доступ к любому элементу строки для чтения и записи, а итерация встречается чаще рекурсии. Однако в XSLT этот алгоритм придется реализовывать рекурсивно, а роскоши манипулировать переменными «на месте» вы лишены (см. пример 2.1).

Пример 2.1. Очень плохая реализация reverse

```
<xsl:template name="reverse">
    <xsl:param name="input"/>
    <xsl:variable name="len" select="string-length($input)"/>
    <xsl:choose>
           <!-- Строки длиной меньше 2 обращаются тривиально -->
           <xsl:when test="$len &lt; 2">
```

```
                <xsl:value-of select="$input"/>
            </xsl:when>
            <!-- Строки длиной 2 также обращаются тривиально -->
            <xsl:when test="$len = 2">
                <xsl:value-of select="substring($input,2,1)"/>
                <xsl:value-of select="substring($input,1,1)"/>
            </xsl:when>
            <xsl:otherwise>
                <!-- Конкатенировать last + reverse(middle) + first -->
                <xsl:value-of select="substring($input,$len,1)"/>
                <xsl:call-template name="reverse">
                    <xsl:with-param name="input"
                        select="substring($input,2,$len - 2)"/>
                </xsl:call-template>
                <xsl:value-of select="substring($input,1,1)"/>
            </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Основной недостаток этого решения в том, что оно годится лишь для очень коротких строк. Проблема возникает из-за того, что рекурсия здесь не хвостовая (см. врезку «Хвостовая рекурсия»). Многие процессоры XSLT (в частности, Saxon) оптимизированы для выполнения хвостовой рекурсии, поэтому код следует структурировать так, чтобы эта оптимизация стала возможной. В примере 2.2 это решение переделано так, чтобы переместить рекурсию в хвост. Для этого при каждом рекурсивном вызове в начало строки перемещается только последний символ. При таком подходе рекурсию можно оптимизировать.

Пример 2.2. Неэффективная реализация с применением хвостовой рекурсии

```
<xsl:template name="reverse">
  <xsl:param name="input"/>
  <xsl:variable name="len" select="string-length($input)"/>
  <xsl:choose>
    <!-- Строки длиной меньше 2 обращаются тривиально -->
    <xsl:when test="$len &lt; 2">
        <xsl:value-of select="$input"/>
    </xsl:when>
    <!-- Строки длиной 2 также обращаются тривиально -->
    <xsl:when test="$len = 2">
        <xsl:value-of select="substring($input,2,1)"/>
        <xsl:value-of select="substring($input,1,1)"/>
    </xsl:when>
    <!-- Конкатенировать last + reverse(rest) -->
```

```
    <xsl:otherwise>
      <xsl:value-of select="substring($input,$len,1)"/>
          <xsl:call-template name="reverse">
            <xsl:with-param name="input"
                            select="substring($input,1,$len - 1)"/>
       </xsl:call-template>
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

При таком изменении мы избегаем переполнения стека, но для длинных строк алгоритм все равно работает неэффективно. Во-первых, отметим, что на каждом шаге перемещается лишь один символ. Во-вторых, каждый рекурсивный вызов должен обрабатывать строку, длина которой лишь на единицу меньше исходной. Для очень длинных строк это может привести к непосильной нагрузке на подсистему управления памятью процессора XSLT. При редактировании этого рецепта Джени Теннисон отметила, что есть еще один способ ввести в решение хвостовую рекурсию – передавать оставшуюся строку (reverse) и `$len` в качестве параметров шаблона. В общем случае это хорошая стратегия, позволяющая добиться хвостовой рекурсии. Однако здесь она лишь позволяет слегка поправить дело, но не достигает той же эффективности, как вариант, приведенный в решении.

При реализации любого рекурсивного алгоритма следует стремиться структурировать его так, чтобы при каждом рекурсивном вызове сложность задачи уменьшалась хотя бы вдвое. Это позволяет быстрее добраться до «дна» рекурсии. Следуя этой рекомендации, мы приходим к реализации шаблона `reverse`, показанной в примере 2.3.

Пример 2.3. Эффективная (но не идеальная) реализация

```
<xsl:template name="reverse">
    <xsl:param name="input"/>

    <xsl:variable name="len" select="string-length($input)"/>
    <xsl:choose>
           <xsl:when test="$len &lt; 2">
               <xsl:value-of select="$input"/>
           </xsl:when>
           <xsl:otherwise>
               <xsl:variable name="mid" select="floor($len div 2)"/>
               <xsl:call-template name="reverse">
                    <xsl:with-param name="input"
                         select="substring($input,$mid+1,$mid+1)"/>
               </xsl:call-template>
               <xsl:call-template name="reverse">
```

```
                <xsl:with-param name="input"
                    select="substring($input,1,$mid)"/>
            </xsl:call-template>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Это первое решение, которое я нашел, и оно приемлемо работает даже для длинных строк (1000 символов и более). Преимущество еще и в том, что оно короче варианта, приведенного в разделе «Решение». Единственная разница заключается в том, что здесь тривиальными считаются только строки длиной 0 или 1. Чуть более быстрая реализация уменьшает число рекурсивных вызовов вдвое за счет непосредственного обращения также строк длиной 2.

Во всех показанных выше реализациях выполняется одно и то же число конкатенаций, и я не думаю, что существует способ уменьшить это число, оставаясь в рамках XSLT. Однако, как показывает тестирование на строках длиной 1000, наилучшее решение быстрее наихудшего примерно в 5 раз. Лучшее и следующее за ним решение отличаются по быстродействию только в 1.3 раза.

ХВОСТОВАЯ РЕКУРСИЯ

Рекурсивный вызов называется *хвостовым*, если возвращенное им значение сразу же возвращается в виде значения вызывающей функции. Слово «хвостовой» намекает на то, что рекурсивный вызов находится в конце функции. Важность хвостовой рекурсии обусловлена тем, что ее можно реализовать более эффективно, чем рекурсию общего вида. В общем случае для рекурсивного вызова нужно подготовить новый кадр стека, в котором будут храниться локальные переменные и другие необходимые данные. Поэтому, если объем входных данных велик, то стек может быстро исчерпаться. Что же касается хвостовой рекурсии, то распознающий ее процессор XSLT способен самостоятельно преобразовать рекурсию в итерацию.

XSLT 2.0

В XSLT 1.0 манипуляции со строками производятся на уровне подстрок, поскольку не существует способа опуститься до уровня символов Unicode. В решении для XSLT 2.0 применяются функции `string-to-codepoints` и `codepoints-to-string`, которые, вероятно, в большинстве реализации работают быстрее, так как во внутреннем представлении строка – это просто массив целых чисел в кодировке Unicode.

2.7. Замена текста

Задача

Требуется заменить все вхождения заданной подстроки другой строкой.

Решение

XSLT 1.0

Следующий рекурсивный шаблон заменяет все вхождения искомой строки на строку замены.

```
<xsl:template name="search-and-replace">
    <xsl:param name="input"/>
    <xsl:param name="search-string"/>
    <xsl:param name="replace-string"/>
    <xsl:choose>
        <!-- Смотрим, содержит ли входная строка искомую -->
        <xsl:when test="$search-string and
                        contains($input,$search-string)">
        <!-- Если да, конкатенируем подстроку, предшествующую искомой,
        со строкой замены, и со строкой, являющейся результатом
        рекурсивного применения шаблона к оставшейся подстроке -->
            <xsl:value-of
                 select="substring-before($input,$search-string)"/>
            <xsl:value-of select="$replace-string"/>
            <xsl:call-template name="search-and-replace">
                <xsl:with-param name="input"
                select="substring-after($input,$search-string)"/>
                <xsl:with-param name="search-string"
                select="$search-string"/>
                <xsl:with-param name="replace-string"
                      select="$replace-string"/>
            </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
             <!-- Больше вхождений искомой строки нет, поэтому возвращаем
             текущую входную строку -->
             <xsl:value-of select="$input"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Если вы хотите заменять только слова целиком, то следует проверять, что непосредственно до и после искомой строки находятся символы, принадлежащие классу разделителей слов. Мы будем считать, что разделителями являются символы, хранящиеся в переменной `$punc`, а также все символы пропуска.

```
<xsl:template name="search-and-replace-whole-words-only">
    <xsl:param name="input"/>
    <xsl:param name="search-string"/>
    <xsl:param name="replace-string"/>
```

```
    <xsl:variable name="punc"
     select="concat('.,;:()[]!?$@&amp;&quot;',&quot;&apos;&quot;)"/>
       <xsl:choose>
        <!-- Смотрим, содержит ли входная строка искомую -->
        <xsl:when test="contains($input,$search-string)">
        <!-- Если да, проверяем, что до и после нее находятся
        разделители слов -->
            <xsl:variable name="before"
              select="substring-before($input,$search-string)"/>
            <xsl:variable name="before-char"
              select="substring(concat(' ',$before),
                                string-length($before) + 1,1)"/>
            <xsl:variable name="after"
              select="substring-after($input,$search-string)"/>
            <xsl:variable name="after-char"
              select="substring($after,1,1)"/>
            <xsl:value-of select="$before"/>
            <xsl:choose>
              <xsl:when test="(not(normalize-space($before-char)) or
                        contains($punc,$before-char)) and
                     (not(normalize-space($after-char)) or
                        contains($punc,$after-char))">
                 <xsl:value-of select="$replace-string"/>
               </xsl:when>
               <xsl:otherwise>
                 <xsl:value-of select="$search-string"/>
               </xsl:otherwise>
            </xsl:choose>
            <xsl:call-template name="search-and-replace-whole-words-only">
              <xsl:with-param name="input" select="$after"/>
              <xsl:with-param name="search-string" select="$search-string"/>
              <xsl:with-param name="replace-string" select="$replace-string"/>
            </xsl:call-template>
       </xsl:when>
    <xsl:otherwise>
       <!-- Больше вхождений искомой строки нет, поэтому возвращаем
       текущую входную строку -->
            <xsl:value-of select="$input"/>
       </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Обратите внимание на то, как переменная `$punc` строится с помощью функции `concat()`, чтобы в нее вошли символы одиночной и двойной кавычек.

Никак по-другому это сделать невозможно, поскольку ни XPath, ни XSLT, в отличие от языка C, не позволяют экранировать специальные символы с помощью обратной косой черты (\). В XPath 2.0 кавычку можно ввести в текст программы, записав ее два раза подряд.

XSLT 2.0

Функциональность шаблона `search-and-replace` в версии 2.0 встроена в функцию `replace()`. Функциональность шаблона `search-and-replace-whole-words-only` можно имитировать с помощью регулярных выражений для сопоставления со словами:

```
<xsl:function name="ckbk:search-and-replace-whole-words-only">
    <xsl:param name="input" as="xs:string"/>
    <xsl:param name="search-string" as="xs:string"/>
    <xsl:param name="replace-string" as="xs:string"/>
    <xsl:sequence select="replace($input, concat('(^|\W)',
$search-string,'(\W|$)'),concat('$1',$replace-string,'$2'))"/>
</xsl:function>
```

Во многих реализациях регулярных выражений для сопоставления с границей слова предусмотрен метасимвол `\b`, но в XPath 2.0 он не поддерживается.

Здесь мы строим регулярное выражение, окружая строку `$search-string` конструкциями `(^|\W)` и `(\W|$)`, где `\W` означает «не `\w`» или «не символ, входящий в состав слова». Метасимволы `^` и `$` учитывают случай, когда слово находится в начале или в конце строки. Мы должны также вернуть сопоставленный символ назад в текст, воспользовавшись ссылками на запомненные группы `$1` и `$2`.

Функция `replace()` позволяет больше, чем в решении для XPath 1.0, так как она пользуется регулярными выражениями и может запоминать отдельные сопоставленные части и подставлять их в строку замены с помощью псевдопеременных `$1`, `$2` и т.д. Мы изучим функцию `replace()` более детально в рецепте 2.10.

Обсуждение

Поиск и замена – типичная задача обработки текста. Представленное выше решение – это самая прямолинейная реализация, написанная на чистом XSLT. У читателя может возникнуть мысль, что производительность такого решения недостаточна. Ведь для каждого выхождения искомой строки вызываются функции `contains()`, `substring-before()` и `substring-after()`. Вполне вероятно, что каждая из этих функций повторно просматривает всю входную строку в поисках искомой. И, стало быть, при таком подходе выполняется на два поиска больше, чем необходимо. Немного поразмыслив, вы можете найти решения, показанные в примерах 2.4 и 2.5, которые на первый взгляд представляются более эффективными.

Пример 2.4. Использование временной строки в неудачной попытке улучшить производительность поиска и замены

```
<xsl:template name="search-and-replace">
    <xsl:param name="input"/>
    <xsl:param name="search-string"/>
    <xsl:param name="replace-string"/>
    <!-- Найти подстроку, предшествующую искомой строке,
    и сохранить ее в переменной -->
    <xsl:variable name="temp"
        select="substring-before($input,$search-string)"/>
    <xsl:choose>
        <!-- Если $temp не пуста или входная строка начинается с искомой
        подстроки, то необходимо произвести замену. Тем самым мы
        избегаем вызова функции contains(). -->
        <xsl:when test="$temp or starts-with($input,$search-string)">
             <xsl:value-of select="concat($temp,$replace-string)"/>
             <xsl:call-template name="search-and-replace">
                  <!-- Вызова substring-after избегаем за счет
                  использования длины temp и искомой строки
                  для извлечения остатка строки в рекурсивном вызове. -->
                  <xsl:with-param name="input"
                  select="substring($input,string-length($temp)+
                         string-length($search-string)+1)"/>
                  <xsl:with-param name="search-string"
                         select="$search-string"/>
                  <xsl:with-param name="replace-string"
                         select="$replace-string"/>
              </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
              <xsl:value-of select="$input"/>
          </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Пример 2.5. Использование временного целого в неудачной попытке улучшить производительность поиска и замены

```
<xsl:template name="search-and-replace">
    <xsl:param name="input"/>
    <xsl:param name="search-string"/>
    <xsl:param name="replace-string"/>
    <!-- Найти длину подстроки, предшествующей искомой строке,
    и сохранить ее в переменной -->
```

```
    <xsl:variable name="temp"
    select="string-length(substring-before($input,$search-string))"/>
    <xsl:choose>
    <!-- Если $temp не равно 0 или входная строка начинается
    с искомой подстроки, то необходимо произвести замену.
    Тем самым мы избегаем вызова функции contains(). -->
           <xsl:when test="$temp or starts-with($input,$search-string)">
               <xsl:value-of select="substring($input,1,$temp)"/>
               <xsl:value-of select="$replace-string"/>
                   <!-- Вызова substring-after избегаем за счет
                   использования temp и длины искомой строки для
                   извлечения остатка строки в рекурсивном вызове. -->
               <xsl:call-template name="search-and-replace">
                  <xsl:with-param name="input"
                      select="substring($input,$temp+
                            string-length($search-string)+1)"/>
                  <xsl:with-param name="search-string"
                      select="$search-string"/>
                  <xsl:with-param name="replace-string"
                      select="$replace-string"/>
                  </xsl:call-template>
           </xsl:when>
           <xsl:otherwise>
                  <xsl:value-of select="$input"/>
           </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Идея обоих вариантов одна и та же: если запомнить, где функция `substring-before()` нашла соответствие, то можно воспользоваться этой информацией для того, чтобы не вызывать функции `contains()` и `substring-after()`. Но мы вынуждены обращаться к функции `starts-with()`, чтобы выделить случай, когда `substring-before()` возвращает пустую строку; такое может случиться, если искомая строка отсутствует или исходная строка начинается с искомой. Впрочем, `starts-with()`, вероятно, работает быстрее, чем `contains()`, поскольку ей не нужно просматривать больше символов, чем содержится в искомой строке. Второй вариант отличается от первого предположением, что сохранение целочисленного смещения может оказаться эффективнее сохранения подстроки целиком.

Увы, ни одна из этих оптимизаций не дает никакого выигрыша при использовании процессора XSLT Xalan. Более того, при некоторых входных данных реализации Saxon и XT показывают *на порядок* большее время работы! Столкнувшись с этим парадоксальным результатом, я сначала предположил, что использование переменной `$temp` в рекурсивном вызове как-то препятствует оптимизации хвостовой рекурсии в Saxon (см. рецепт 2.6). Однако, экспериментируя с длинными входными строками, в которых искомая строка встречается много раз, я не сумел вызвать переполнение

стека. Тогда я заподозрил, что по какой-то причине функция `substring()` в XSLT работает медленнее, чем `substring-before()` и `substring-after()`. Майкл Кэй, автор реализации Saxon, указал, что `substring()` действительно работает медленно из-за сложных правил, которые приходится поддерживать, в том числе округления аргументов с плавающей точкой, обработки особых случаев, когда начальная или конечная точки оказываются за границами строки, и вопросов, связанных с суррогатными парами Unicode. Напротив, функции `substring-before()` и `substring-after()` гораздо лучше транслируются на язык Java.

Отсюда следует извлечь урок: оптимизация – дело непростое, особенно в XSLT, когда имеются существенные различия между реализациями, а в новых версиях авторы стараются применить дополнительные оптимизации. Если вы не готовы часто профилировать программу, то лучше ограничиться простыми решениями. К числу достоинств простых решений можно отнести и то, что, скорее всего, они будут вести себя одинаково в разных реализациях XSLT.

2.8. Преобразование регистра

Задача

Требуется преобразовать строку, содержащую символы верхнего регистра, в нижний или наоборот.

Решение

XSLT 1.0

Воспользуйтесь функцией `translate()`. Так, следующий код преобразует заглавные буквы в строчные:

```
translate($input, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'abcdefghijklmnopqrstuvwxyz')
```

А этот код выполняет обратное преобразование:

```
translate($input, 'abcdefghijklmnopqrstuvwxyz', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ')
```

XSLT 2.0

Воспользуйтесь функциями `upper-case()` и `lower-case()`:

```
upper-case($input)
lower-case($input)
```

Обсуждение

Конечно, этот рецепт тривиален. Но я включил его, чтобы воспользоваться случаем и обсудить недостатки решения для XPath 1.0. Преобразование регистров – тривиальная задача лишь до тех пор, пока текст записан в одной локали. В английском тексте вам вряд ли придется столкнуться со специальными символами, которые содержат диакритические знаки, или такими преобразованиями, когда один символ превращается в два. Самый распространенный пример –

немецкий язык, в котором строчная буква Я (β-цет) в верхнем регистре записывается как SS. Во многих современных языках программирования есть функции преобразования регистра, учитывающие локаль, но в XSLT прямой поддержки этой идеи нет. Это печально, поскольку другие средства поддержки интернационализации в XSLT все же включены.

Небольшого улучшения можно добиться, определив для каждого преобразования отдельные XML-компоненты, как в следующем примере:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stylesheet [
    <!ENTITY UPPERCASE "ABCDEFGHIJKLMNOPQRSTUVWXYZ">
    <!ENTITY LOWERCASE "abcdefghijklmnopqrstuvwxyz">
    <!ENTITY UPPER_TO_LOWER " '&UPPERCASE;' , '&LOWERCASE;' ">
    <!ENTITY LOWER_TO_UPPER " '&LOWERCASE;' , '&UPPERCASE;' ">
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="/">
    <xsl:variable name="test"
        select=" 'The rain in Spain falls mainly in the plain' "/>
    <output>
        <lowercase>
           <xsl:value-of
               select="translate($test,&UPPER_TO_LOWER;)"/>
        </lowercase>
        <uppercase>
           <xsl:value-of
               select="translate($test,&LOWER_TO_UPPER;)"/>
        </uppercase>
    </output>
    </xsl:template>

</xsl:stylesheet>
```

Эти определения компонентов играют троякую роль. Во-первых, они упрощают перенос таблицы стилей в другую локаль, поскольку изменить нужно лишь определения UPPERCASE и LOWERCASE. Во-вторых, они делают код более компактным, так как отпадает необходимость перечислять все буквы алфавита дважды. В-третьих, наше намерение вызвать функцию translate становится очевидно всякому человеку, читающему код. Некоторые ревнители чистоты могут возразить против переноса третьего параметра translate в макрос, но я думаю, что так код читать проще. Если вы тоже сторонник пуризма, то можете писать так: translate($test, &UPPERCASE;, &LOWERCASE;).

Я не часто встречал употребление компонентов в книгах, посвященных XSLT, но полагаю, что этот прием обладает рядом достоинств. Вообще, одно из преимуществ

записи XSLT-программы в синтаксической нотации XML заключается в том, что вы получаете возможность пользоваться всеми средствами XML, в том числе и определением компонентов. Если вы намереваетесь применять эту технику и планируете написать достаточно много таблиц стилей, то подумайте над тем, чтобы поместить общие определения компонентов во внешний файл и включать их, как показано в примере 2.6. Можно также хранить эти значения в глобальных переменных во внешней таблице стилей и импортировать при необходимости. Такую альтернативу предпочитают многие ветераны XSLT.

Пример 2.6. Файл standard.ent

```
<!ENTITY UPPERCASE "ABCDEFGHIJKLMNOPQRSTUVWXYZ">
<!ENTITY LOWERCASE "abcdefghijklmnopqrstuvwxyz">
<!ENTITY UPPER_TO_LOWER " '&UPPERCASE;' , '&LOWERCASE;' ">
<!ENTITY LOWER_TO_UPPER " '&LOWERCASE;' , '&UPPERCASE;' ">
<!-- прочие... -->
```

Воспользоваться параметрическим компонентом, определенным в файле *standard.ent*, можно, как показано в примере 2.7.

Пример 2.7. Таблица стилей, в которой используется файл standard.ent

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE stylesheet [
    <!ENTITY % standard SYSTEM "standard.ent">
    %standard;
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- ... -->
</xsl:stylesheet>
```

Реализация преобразования регистра, предложенная Стивом Болом (Steve Ball) и работающая практически во всех случаях, достигает этого за счет включения большинства общеупотребительных символов Unicode в строки, записанные в верхнем и нижнем регистре. При этом немецкая буква β обрабатывается специально.

XSLT 2.0

Добавленные в XPath 2.0 функции `upper-case()` и `lower-case()` решают большую часть проблем, связанных с преобразованием регистра в не-английских алфавитах. Единственное исключение касается преобразований с учетом местных особенностей. Лучше не пользоваться этими функциями для сравнения строк без учета регистра. Вместо этого обратитесь к функции `compare()`, указав таблицу сравнения (collation), в которой регистр игнорируется. Пользователь Saxon 8.x найдет информацию о таблицах сравнения на страницах *http://www.saxonica.com/documentation/conformance/ collation-uri.html* и *http://www.saxonica.com/documentation/extensions/instructions/ collation.html*.

См. также

Решение Стива Бола можно найти в «стандартной библиотеке XSLT» по адресу *http://xsltsl.sourceforge.net*.

2.9. Разбиение строки на лексемы

Задача

Требуется разбить строку на лексемы, используя в качестве разделителей один или несколько заданных символов.

Решение

XSLT 1.0

Само решение принадлежит Джени Теннисон (мои лишь комментарии). Каждая лексема возвращается в виде узла, состоящего из элемента `token`, содержащего текст. Если строка разделителей пуста, то по умолчанию исходная строка разбивается на отдельные символы.

```
<xsl:template name="tokenize">
  <xsl:param name="string" select="''" />
  <xsl:param name="delimiters" select="' &#x9;&#xA;'" />
  <xsl:choose>
      <!-- Ничего не делать, если строка пуста -->
    <xsl:when test="not($string)" />

     <!-- Если разделителей нет, строка разбивается на отдельные символы. -->
    <xsl:when test="not($delimiters)">
      <xsl:call-template name="_tokenize-characters">
        <xsl:with-param name="string" select="$string" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="_tokenize-delimiters">
        <xsl:with-param name="string" select="$string" />
        <xsl:with-param name="delimiters" select="$delimiters" />
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="_tokenize-characters">
  <xsl:param name="string" />
  <xsl:if test="$string">
    <token><xsl:value-of select="substring($string, 1, 1)" /></token>
```

```
    <xsl:call-template name="_tokenize-characters">
      <xsl:with-param name="string" select="substring($string, 2)" />
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="_tokenize-delimiters">
  <xsl:param name="string" />
  <xsl:param name="delimiters" />
  <xsl:param name="last-delimit"/>
  <!-- Извлечь разделитель -->
  <xsl:variable name="delimiter" select="substring($delimiters, 1, 1)" />
  <xsl:choose>
      <!-- Если строка разделителей пуста, имеем лексему -->
    <xsl:when test="not($delimiter)">
      <token><xsl:value-of select="$string"/></token>
    </xsl:when>
      <!-- Если строка содержит хотя бы один разделитель, мы должны
      разбить ее -->
    <xsl:when test="contains($string, $delimiter)">
      <!-- Если строка начинается с разделителя, обрабатывать
      предшествующую подстроку не нужно -->
    <xsl:if test="not(starts-with($string, $delimiter))">
      <!-- Обрабатываем часть, предшествующую текущему разделителю, -->
      <!-- пробуя следующий разделитель. Если следующего нет, то первая
      <!-- проверка в этом шаблоне выделяет лексему -->
    <xsl:call-template name="_tokenize-delimiters">
     <xsl:with-param name="string"
                     select="substring-before($string, $delimiter)" />
     <xsl:with-param name="delimiters"
                     select="substring($delimiters, 2)" />
    </xsl:call-template>
   </xsl:if>
     <!-- Обрабатываем часть, следующую за разделителем, применяя
     текущий разделитель -->
   <xsl:call-template name="_tokenize-delimiters">
     <xsl:with-param name="string"
                     select="substring-after($string, $delimiter)" />
     <xsl:with-param name="delimiters" select="$delimiters" />
   </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <!-- Текущий разделитель не встречается, поэтому переходим
      к следующему -->
```

```
    <xsl:call-template name="_tokenize-delimiters">
         <xsl:with-param name="string"
                         select="$string" />
         <xsl:with-param name="delimiters"
                         select="substring($delimiters, 2)" />
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

XSLT 2.0

Воспользуйтесь встроенной функцией `tokenize()`, которая рассматривается в рецепте 2.11.

Обсуждение

Выделение лексем – типичная задача обработки текста. В языках, где есть развитый аппарат регулярных выражений, она решается тривиально. В этом отношении такие языки, как Perl, Python, JavaScript и Tcl пока превосходят XSLT. Однако, как показано в этом рецепте, разбиение на лексемы можно выполнить, даже не выходя за пределы чистого XSLT. Если вы готовы прибегнуть к расширениям, то можете реализовать низкоуровневые операции со строками на каком-нибудь другом языке.

Если же вам больше нравится подход XSLT, но ваш процессор не оптимизирует хвостовую рекурсию, то в шаблоне `_tokenize-characters` можно воспользоваться алгоритмом «разделяй и властвуй»:

```
<xsl:template name="_tokenize-characters">
  <xsl:param name="string" />
  <xsl:param name="len" select="string-length($string)"/>
  <xsl:choose>
     <xsl:when test="$len = 1">
     <token><xsl:value-of select="$string"/></token>
     </xsl:when>
     <xsl:otherwise>
    <xsl:call-template name="_tokenize-characters">
    <xsl:with-param name="string"
                    select="substring($string, 1, floor($len div 2))" />
    <xsl:with-param name="len" select="floor($len div 2)"/>
  </xsl:call-template>
    <xsl:call-template name="_tokenize-characters">
     <xsl:with-param name="string"
                     select="substring($string, floor($len div 2) + 1)" />
     <xsl:with-param name="len" select="ceiling($len div 2)"/>
```

```
      </xsl:call-template>
       </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

В главе 12 показано, как обратиться к регулярным выражениям JavaScript, если ваш процессор XSLT допускает расширения на этом языке. В языке Java также имеется готовый класс для разбиения строки на лексемы (`java.util.StringTokenizer`).

2.10. Как обойтись без регулярных выражений

Задача

Вы хотели бы выполнять операции, подобные регулярным выражениям в XSLT 1.0, но не склонны прибегать к нестандартным расширениям.

Решение

Некоторые типичные сопоставления, свойственные регулярным выражениям, можно эмулировать в XPath 1.0. В таблице 2.1 приведены регулярные выражения в синтаксисе Perl и эквивалентные им конструкции XSLT/XPath. Одиночный символ «C» обозначает любой заданный пользователем символ, а строка «abc» – произвольную строку ненулевой длины.

Таблица 2.1. *Сопоставление с регулярными выражениями*

```
$string =~ /^C*$/   translate($string,'C','') = ''
$string =~ /^C+$/   $string and translate($string,'C','') = ''
$string =~ /C+/     contains($string,'C')
$string =~ /C{2,4}/ contains($string,'CC') and not contains($string,'CCCC')
$string =~ /^abc/   starts-with($string,'abc')
$string =~ /abc$/   substring($string, string-length($string) -
                    string-length('abc') + 1) = 'abc'
$string =~ /abc/    contains($string,'abc')
$string =~ /^[^C]*$/ translate($string,'C','') = $string
$string =~ /^\s+$/  not(normalize-space($string))
$string =~ /\s/     translate(normalize-space($string), ' ', '') != $string
$string =~ /^\S+$/  translate(normalize-space($string), ' ', '') = $string
```

Обсуждение

Если нужно добиться краткости и мощи, то ничто не может сравниться с хорошим механизмом регулярных выражений. Однако многие простые операции сопоставления можно эмулировать с помощью не таких элегантных, но достаточно эффективных выражений XPath. При этом часто используется функция `translate()`, которая удаляет лишние символы, после чего сопоставление становится простой проверкой на равенство. Еще одно полезное применение `translate()` – это возможность подсчитать число вхождений конкретного символа

или символов из заданного набора. Например, следующий код вычисляет, сколько раз в строке встречаются цифры:

```
string-length(translate($string,
           translate($string, '0123456789', ''), ''))
```

Если вам непонятно, как работает этот код, обратитесь к рецепту 2.3. Можно написать и так:

```
string-length($string) -
string-length(translate($string, '0123456789', ''))
```

Здесь вместо `translate()` мы прибегаем к дополнительному вызову `string-length()` с последующим вычитанием. Возможно, это окажется чуть быстрее.

Важным отличием этих выражений XPath от их аналогов в Perl является то, что в Perl в качестве побочного эффекта сопоставления с образцом устанавливаются еще значения специальных переменных. С их помощью можно обрабатывать строки так, как XSLT и не снилось. Если кто-нибудь попробует «поженить» Perl и XSLT, придумав гибридный язык, я хотел бы стать одним из первых альфа-тестеров!

Но радует то, что XPath 2.0 поддерживает регулярные выражения, и я рассмотрю это давно ожидаемое дополнение в рецепте 2.1.

2.11. Использование регулярных выражений

Задача

Вы слыхали о том, что регулярные выражения – это новый мощный инструмент, появившийся в XSLT 2.0, но не знаете, как применить эту мощь для своих нужд.

Решение

Сопоставление с текстовыми образцами

Важнейшее применение регулярные выражения находят при сопоставлении с текстовыми образцами. Воспользовавшись функцией `matches()` в правиле шаблона, можно выполнить сопоставление с текстом узла:

```
<!-- -->

<!-- Дата в формате May 3, 1964 -->
<xsl:template  match="birthday[matches(.,'^[A-Z][a-z]+\s[0-9]+,\s[0-9]+$')]">
    <!-- ... -->
</xsl:template>

<!-- Дата в формате 1964-05-03 -->
<xsl:template match="birthday[matches(.,'^[0-9]+-[0-9]+-[0-9]+$')]">
```

```
    <!-- ... -->
</xsl:template>

<!-- Дата в формате 3 May 1964 -->
<xsl:template match="birthday[matches(.,'^[0-9]+\s[A-Z][a-z]+\s[0-9]+$')]">
    <!-- ... -->
</xsl:template>
```

Можно также выполнять сопоставление в командах `xsl:if` и `xsl:choose`:

```
<xsl:choose>
    <xsl:when test="matches($date,'^[A-Z][a-z]+\s[0-9]+,\s[0-9]+$')">
    </xsl:when>
    <xsl:when test="matches($date,'^[0-9]+-[0-9]+-[0-9]+$')">
    </xsl:when>
    <xsl:when test="matches($date,'^[0-9]+\s[A-Z][a-z]+\s[0-9]+$')">
    </xsl:when>
</xsl:choose>
```

Выделение лексем в стилизованном тексте

Часто регулярные выражения применяются для разбиения строки на лексемы:

```
(: Выделить из даты в формате ISO (YYYY-MM-DDhh:mm:ss) год, месяц и день,
представив их в виде последовательности :)
tokenize($date, '-')

(: Выделить из даты в формате ISO (YYYY-MM-DDhh:mm:ss) год, месяц, день,
часы, минуты и секунды представив их в виде последовательности :)
tokenize($date, '-|T|:')

(: Разбить предложение на слова :)
tokenize($text, '\W+')
```

Замена и расширение текста

Есть два способа применения функции `replace()`, входящей в состав XPath.

Во-первых, можно просто заменить найденные образцы другим текстом. Иногда образец заменяется пустой строкой (''), поскольку найденный текст нужно удалить.

```
(: Заменить день месяца в дате в формате ISO на 01 :)
replace($date, '\d\d$', '01')

(: Убрать из даты в формате ISO все компоненты, кроме года :)
replace($date, '-\d\d-\d\d$', '')
```

При втором способе использования `replace` вы вставляете текст в место, где произошло сопоставление с образцом, не изменяя самого сопоставленного фрагмента. Возможно, тот факт, что функция, называемая `replace` (заменить), может

выполнять вставку текста, и противоречит интуиции, но именно такого результата позволяют добиться обратные ссылки.

```
(: Вставить пробел после знаков препинания, за которыми не следует пробел :)
replace($text, '([,;:])\S', '$1 ')
```

Анализ текста для преобразования в формат XML

Еще более мощной, чем функции `tokenize()` и `replace()`, является появившаяся в XSLT 2.0 команда `xsl:analyze-string`. Она позволяет не просто выполнять подстановки в тексте, но строить на основе текста XML-содержимое. Рецепты использования `xsl:analyze-string` см. в главе 6.

Обсуждение

Регулярные выражения – это настолько богатый и мощный инструмент обработки текста, что им можно было бы посвятить целую книгу. И такие люди нашлись. Книга Джеффри Фридла «Регулярные выражения» (издательство «Питер», 2003) – классическое сочинение на эту тему, я ее настоятельно рекомендую.

Важность регулярных выражений проистекает из умения сопоставлять текст с образцами. Интересно, что сопоставление с образцами – это и основа мощи языка XSLT. Но если XSLT идеально приспособлен для поиска образцов в структуре XML-документа, то регулярные выражения оптимизированы для работы с произвольным текстом. Однако язык регулярных выражений сложнее языка выражений XPath, используемого внутри XSLT. Это неизбежно просто потому, что произвольному тексту недостает древовидной структуры, характерной для XML.

Ключ к овладению регулярными выражениями – практика и разумное заимствование идей из примеров выражений, написанных другими людьми. Помимо книга Фридла, примеры можно найти в книгах по языку Perl и на сайте RegExLib.com (*http://regexlib.com*).

2.12. Расширения EXSLT для работы со строками

Задача

У вас есть основательные причины воспользоваться дополнительными функциями для обработки строк, но беспокоит вопрос переносимости.

Решение

Не исключено, что ваш процессор XSLT уже реализует строковые функции, определенные сообществом EXSLT (*http://www.exslt.org*). На момент работы над этой книгой в их число входили следующие функции.

```
node-set str::tokenize(string input, string delimiters?)
```

Функция `str::tokenize` разбивает строку на лексемы и возвращает набор узлов, состоящий из элементов token, каждый из которых содержит текст одной лексемы.

Первый аргумент – подлежащая разбиению строка. Второй аргумент – строка, содержащая символы-разделители. Строка, переданная в первом аргументе, разбивается в местах, где встречается любой символ-разделитель.

Если второй аргумент опущен, по умолчанию подразумевается строка `	

 ` (то есть все символы пропуска).

Если второй аргумент – пустая строка, то функция возвращает набор элементов token, каждый из которых содержит по одному символу.

```
node-set str::replace(string, object search, object replace)
```

Функция `str:replace` заменяет все вхождения искомых подстрок в исходную строку набором узлов замены, создавая набор узлов.

Первый аргумент – строка, в которой ищутся заменяемые подстроки.

Второй аргумент – объект, описывающий список искомых строк. Если он представляет собой набор узлов, то список искомых строк получается путем преобразования каждого узла в строку с помощью функции `string()`, причем порядок строк в списке определяется порядком узлов в документе. Если второй аргумент – не набор узлов, то он преобразуется в строку функцией `string()`, так что список искомых строк будет содержать всего одну строку.

Третий аргумент – объект, представляющий список узлов замены. Если это набор узлов, то список состоит из этих узлов, перечисленных в порядке документа. Если же это не набор узлов, то список узлов замены состоит из единственного текстового узла, строковое значение которого получается преобразованием третьего аргумента в строку функцией `string()`.

```
string str::padding(number, string?)
```

Функция `str::padding` создает строку заполнения заданной длины.

Первый аргумент – длина требуемой строки заполнения.

Второй аргумент – строка, на основе которой создается строка заполнения. Она повторяется столько раз, сколько необходимо для получения строки, длина которой задана первым аргументом. Если эта строка содержит более одного символа, то результат может быть обрезан для получения требуемой длины. Если второй аргумент опущен, по умолчанию подразумевается пробел (`" "`). Если второй аргумент – пустая строка, то `str::padding` возвращает пустую строку.

```
string str::align(string, string, string?)
```

Функция `str::align` выравнивает одну строку в границах другой.

Первый аргумент – строка, подлежащая выравниванию. Второй аргумент – строка заполнения, внутри которой выравнивается первая строка.

Если первая строка короче второй, то некоторые символы в строке заполнения заменяются символами из выравниваемой строки. Какие именно символы заменяются, зависит от значения третьего аргумента, определяющего способ выравнивания. Он может принимать значения `left`, `right` или `center`. Если третий аргумент опущен или не совпадает с одним из указанных значений, по умолчанию подразумевается `left`.

При выравнивании по левому краю диапазон символов, заменяемых выравниваемой строкой, начинается с первого символа строки заполнения. При выравнивании по правому краю диапазон символов, заменяемых выравниваемой строкой, заканчивается последним символом строки заполнения. При выравнивании по центру диапазон символов, заменяемых выравниваемой строкой, находится в середине строки заполнения, то есть число не замененных символов слева и справа от этого диапазона одинаково или слева на один символ меньше, чем справа.

Если выравниваемая строка длиннее строки заполнения, то она обрезается до той же длины, что строка заполнения, и возвращается получившийся результат.

```
string str::encode-uri(string)
```

Функция `str:encode-uri` возвращает закодированный URI. При этом следующие символы не подвергаются кодированию: «:», «/», «;» и «?».

При кодировании URI небезопасные и зарезервированные символы заменяются строкой, которая начинается с символа «%», за которым следуют две шестнадцатеричные цифры (0-9, A-F), представляющие значение символа в кодировке ISO Latin 1.

```
string str::decode-uri(string)
```

Функция `str:decode-uri` раскодирует строку, содержащую закодированный URI. См. описание функции `str:encode-uri`.

```
node-set str::concat(node-set)
```

Функция `str:concat` принимает набор узлов и возвращает строку, получающуюся в результате конкатенации строковых значений этих узлов. Если набор узлов пуст, возвращается пустая строка.

```
node-set str::split(string, string?)
```

Функция `str:split` разбивает строку и возвращает набор узлов, состоящий из элементов token, каждый из которых содержит одну лексему. Первый аргумент – подлежащая разбиению строка, второй – строка-образец. Первая строка разбивается в местах, где обнаруживается текст, сопоставившийся с образцом.

Если второй аргумент опущен, по умолчанию принимается строка `&#x20` (пробел).

Если второй аргумент – пустая строка, то функция возвращает набор элементов token, каждый из которых содержит ровно один символ.

Обсуждение

Если вы пользуетесь строковыми функциями EXSLT, то переносимость не гарантируется, поскольку в настоящий момент ни одна реализация XSLT не поддерживает их все. Более того, на сайте EXSLT говорится, что для некоторых функций пока нет вообще ни одной реализации. Сообщество EXSLT компенсирует это, предоставляя, когда возможно, реализации на чистом XSLT, JavaScript или MSXML.

Убедительной причиной пользоваться расширениями EXSLT является тот факт, что члены этого коллектива играют весьма заметную роль в сообществе XSLT, поэтому многие реализации равно или поздно будут поддерживать большую часть предлагаемых расширений. Не исключено также, что их работа хотя бы отчасти войдет в стандарт следующей версии XSLT.

См. также

В карманном справочнике по XSLT 1.0 раздел, относящийся к EXSLT, содержит примечания, в которых расширения сравниваются с функциями XPath/XSLT 2.0, выполняющими аналогичное или в точности такое же действие.

Глава 3. Математические операции над числами

Арифметика – это умение считать до 20, не снимая ботинок.
Микки Маус

3.0. Введение

В главе 2 мы сетовали на отсутствие развитых средств обработки строк в XSLT 1.0. А уж что касается математических вычислений, XSLT и вовсе ничем не лучше Микки Мауса! В XSLT 1.0 имеются средства для выполнения основных арифметических операций, подсчета и суммирования значений и форматирования чисел. Вся остальная математика – на откуп вашей изобретательности. К счастью, как и в случае строк, рекурсия позволяет совершать подвиги на ниве математики с относительно небольшими усилиями.

В XPath/XSLT 2.0 добавлен ряд наиболее очевидных математических функций, в том числе `abs()`, `avg()`, `max()`, `min()`, `round-to-half-even()`. Кроме того, появились числовые типы (`xs:integer`, `xs:double` и другие типы данных, определенные в спецификации XML Schema), тогда как в XSLT 1.0 единственным числовым типом было число одинарной точности с плавающей точкой. В XPath 2.0 наконец-то распознается научная нотация, отсутствие которой в версии 1.0 вызывало значительные неудобства.

Однако ни для первой, ни для второй версии не надейтесь найти в этой главе рецепты умножения матриц или выполнения быстрого преобразования Фурье. Если вам нужны столь сложные математические вычисления над данными, записанными в формате XML, то язык XSLT не для вас. Лучше импортируйте данные в какой-нибудь язык, более благосклонно относящийся к математике, используя XSLT в качестве конвертера, либо обратитесь к интерфейсу SAX или DOM. Впрочем, на странице «Галерея забавных трюков на XSL и XSLT» (*http://www.incrementaldevelopment.com/ xsltrick*) есть несколько любопытных примеров реализации на XSLT таких задач, как вычисление простых чисел или дифференцирование полиномов. Познакомиться с ними поучительно, поскольку это обогатит ваше понимание XSLT. Но в этой главе мы ограничимся рецептами, демонстрирующими решение часто встречающихся математических задач средствами одного лишь XSLT с приемлемой производительностью.

Некоторые примеры, приведенные в начале главы, стоит рассматривать как руководство по применению встроенной в XSLT функциональности. Я включил их, потому что эти средства иногда понимают неправильно.

Многие рассматриваемые рецепты – это реализации функций, определенных в проекте EXSLT. Если на сайте EXSLT.org имеется реализация на чистом XSLT,

мы сначала обсудим ее, а потом поговорим об альтернативных решениях[1]. Реализации на чистом XSLT существуют для всех определенных в EXSLT расширений, кроме тригонометрических функций (`sin`, `cos` и т.д.). Если вы жить не можете без них, то в рецепте 3.5 упомянуто одно из возможных направлений поиска.

Во многих рецептах этой главы исследуются альтернативные решения. Читатели, которых технические подробности не интересуют, могут просто воспользоваться примером, приведенном в разделе «Решение», поскольку оно всегда самое лучшее или, по крайней мере, не хуже прочих.

3.1. Форматирование чисел

Задача

Требуется выводить числа в разных форматах.

Решение

У этой задачи есть два общих решения.

Использование xsl:decimal-format в сочетании с format-number()

Элемент верхнего уровня `xsl:decimal-format` задает именованное правило форматирования, на которое можно ссылаться при вызове функции `format-number()`. У элемента `xsl:decimal-format` много атрибутов, описывающих правила форматирования. В таблице 3.1 описываются все атрибуты, и в скобках для каждого приведено значение по умолчанию.

Таблица 3.1. *Атрибуты элемента xsl:decimal-format*

Атрибут	Назначение
`name`	Необязательное имя правила. Если этот атрибут отсутствует, то правило принимается по умолчанию. Может существовать лишь одно правило по умолчанию, и все имена должны быть уникальны (даже в случае импорта, когда действует система приоритетов).
`decimal-separator (.)`	Символ, отделяющий целую часть от дробной.
`grouping-separator (,)`	Символ, разделяющий группы цифр.
`infinity (Infinity)`	Строка, представляющая бесконечность.
`minus-sign (-)`	Символ, представляющий знак минуса.
`NaN (Nan)`	Строка, представляющая значение «не число».
`percent (%)`	Знак процента.
`per-mille (‰)`	Знак промилле.
zero-digit (0)	Символ, используемый в форматной строке, чтобы показать, где должны находиться начальные или конечные нули. Задание этого символа меняет основание системы счисления. См. пример 3.1.

[1] Реализации EXSLT приведены в том виде, в котором они существовали на момент написания этой книги. Естественно, со временем они могут быть улучшены или отвергнуты по мере выхода в свет новых версий XPath и XSLT.

Таблица 3.1. *Атрибуты элемента xsl:decimal-format (окончание)*

Атрибут	Назначение
`digit (#)`	Символ, используемый в форматной строке, чтобы показать, где должны находиться значащие цифры.
`pattern-separator (;)`	Символ, используемый в форматной строке для разделения положительной и отрицательной подмасок.

В таблице 3.2 описаны аргументы функции `format-number()`.

Таблица 3.2. *Аргументы функции format-number()*

Аргумент	Назначение
`value`	Форматируемое значение.
`format`	Форматная строка (например, #,###.00).
`name` (необязательный)	Имя элемента `xsl:decimal-format`.

Использование xsl:number

Элемент `xsl:number` обычно применяется для нумерации узлов по порядку, но годится и для форматирования чисел. Атрибуты, используемые в этом случае, описаны в таблице 3.3.

Таблица 3.3. *Атрибуты элемента xsl:number*

Имя	Назначение
`value`	Форматируемое число.
`format`	Форматная строка (см. обсуждение).
`lang`	Код языка, определенный атрибутом `xml:lang`.
`letter-value`	Может принимать значение `alphabetic` или `traditional` и служит для различения систем записи чисел.
`grouping-separator`	Одиночный символ, используемый для разделения групп. Например, в США для этой цели применяется запятая.
`grouping-size`	Количество цифр в каждой группе.

В таблице 3.4 показаны, как интерпретируются спецификаторы формата, заданные в форматной строке.

Таблица 3.4. *Примеры применения спецификаторов формата*

Спецификатор	Форматируемое число	Результат
1	1	1
1	99	99
01	1	01
001	1	001
a	1	a
a	10	j
a	27	aa
A	1	A
A	27	AA

Таблица 3.4. *Примеры применения спецификаторов формата (окончание)*

Спецификатор	Форматируемое число	Результат
i	1	i
i	10	x
I	1	I
I	11	XI

Форматная строка представляет собой последовательность спецификаторов формата и знаков пунктуации. Указывать несколько спецификаторов имеет смысл только тогда, когда форматируемое значение содержит несколько чисел.

Обсуждение

С помощью описанного выше механизма можно решить почти любую задачу форматирования чисел.

Форматирование чисел для вывода в несколько колонок с фиксированным числом десятичных знаков

Мы дополним число начальными и конечными нулями, а потом заменим начальные нули пробелами и добавим в конце знак «минус». В результате получится аккуратная таблица с несколькими колонками, выровненными по правому краю, при условии, что для ее вывода на устройство применяется моноширинный шрифт. В примерах 3.1 – 3.3 показаны более привычные способы заполнения. В них иллюстрируется поведение спецификатора форматирования 0.

Пример 3.1. Входные данные

```
<numbers>
  <number>10</number>
  <number>3.5</number>
  <number>4.44</number>
  <number>77.7777</number>
  <number>-8</number>
  <number>1</number>
  <number>444</number>
  <number>1.1234</number>
  <number>7.77</number>
  <number>3.1415927</number>
  <number>10</number>
  <number>9</number>
  <number>8</number>
  <number>7</number>
  <number>666</number>
  <number>5555</number>
  <number>-4444444</number>
```

```
  <number>22.33</number>
  <number>18</number>
  <number>36.54</number>
  <number>43</number>
  <number>99999</number>
  <number>999999</number>
  <number>9999999</number>
  <number>32</number>
  <number>64</number>
  <number>-64.0001</number>
</numbers>
```

Пример 3.2. format-numbers-into-columns.xslt

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

<xsl:variable name="numCols" select="4"/>

<xsl:template match="numbers">
  <xsl:for-each select="number[position() mod $numCols = 1]">
    <xsl:apply-templates
           select=". | following-sibling::number[position() &lt; $numCols]"
           mode="format"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:for-each>
</xsl:template>

<xsl:template match="number" name="format" mode="format">
  <xsl:param name="number" select="." />
  <xsl:call-template name="leading-zero-to-space">
   <xsl:with-param name="input"
                   select="format-number($number,
                                         '0000000.0000 ; 0000000.0000- ')"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="leading-zero-to-space">
  <xsl:param name="input"/>
  <xsl:choose>
    <xsl:when test="starts-with($input,'0')">
    <xsl:value-of select="' '"/>
    <xsl:call-template name="leading-zero-to-space">
      <xsl:with-param name="input" select="substring-after($input,'0')"/>
```

```
    </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$input"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 3.3. Вывод

```
         10.00                3.50               4.40
         77.77               -8.00               1.00
        444.00                1.12               7.77
          3.14               10.00               9.00
          8.00                7.00             666.00
       5555.00          -4444444.00              22.33
         18.00              36.5400              43.00
      99999.00            999999.00         9999999.00
         32.00               64.00              -64.00
```

Форматирование денежных сумм, как принято в США

Пример 3.4. Бухгалтерский формат

```
<xsl:template match="number" name="format" mode="format">
  <xsl:param name="number" select="." />
  <xsl:text> $ </xsl:text>
  <xsl:call-template name="leading-zero-to-space">
    <xsl:with-param name="input"
                    select="format-number($number,
                                          ' 0000000.00 ;(0000000.00)')"/>
  </xsl:call-template>
</xsl:template>

Вывод:
$       10.00  $         3.50  $        4.44
$       77.78  $        (8.00) $        1.00
$      444.00  $         1.12  $        7.77
$        3.14  $        10.00  $        9.00
$        8.00  $         7.00  $      666.00
$     5555.00  $  (4444444.00) $       22.33
$       18.00  $        36.54  $       43.00
$    99999.00  $    999999.00  $  9999999.00
$       32.00  $        64.00  $      (64.00)
```

Форматирование чисел, принятое во многих европейских странах

В примере 3.5 демонстрируется использование именованного формата.

Пример 3.5. Форматирование чисел по-европейски

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings">

<xsl:output method="text" />

<!-- Из рецепта 2.5 -->
<xsl:include href="../strings/str.dup.xslt"/>

<xsl:variable name="numCols" select="3"/>

<xsl:decimal-format name="WesternEurope"
                    decimal-separator="," grouping-separator="."/>

<xsl:template match="numbers">
  <xsl:for-each select="number[position() mod $numCols = 1]">
    <xsl:apply-templates
         select=". | following-sibling::number[position() &lt; $numCols]"
         mode="format"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:for-each>
</xsl:template>

<xsl:template match="number" name="format" mode="format">
  <xsl:param name="number" select="." />
  <xsl:call-template name="pad">
    <xsl:with-param name="string"
                    select="format-number($number,'#.###,00','WesternEurope')"/>
  </xsl:call-template>
</xsl:template>

<xsl:template name="pad">
  <xsl:param name="string"/>
  <xsl:param name="width" select="16"/>
  <xsl:call-template name="str:dup">
    <xsl:with-param name="input" select="' '"/>
    <xsl:with-param name="count" select="$width - string-length($string)"/>
  </xsl:call-template>
  <xsl:value-of select="$string"/>
</xsl:template>
```

```
</xsl:stylesheet>

Вывод:
       10,00              3,50              4,44
       77,78             -8,00              1,00
      444,00              1,12              7,77
        3,14             10,00              9,00
        8,00              7,00            666,00
    5.555,00     -4.444.444,00             22,33
       18,00             36,54             43,00
   99.999,00        999.999,00      9.999.999,00
       32,00             64,00            -64,00
```

Запись числа римскими цифрами

В примере 3.6 элемент `xsl:number` применяется для нумерации строк и колонок римскими цифрами.

Пример 3.6. Нумерация римскими цифрами

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings" >
    <xsl:output method="text" />

<xsl:include href="../strings/str.dup.xslt" />

<xsl:variable name="numCols" select="3"/>

<xsl:template match="numbers">
  <xsl:for-each select="number[position() &lt;= $numCols]">
    <xsl:text>            </xsl:text>
    <xsl:number value="position()" format="I"/><xsl:text>   </xsl:text>
  </xsl:for-each>
    <xsl:text>&#xa;     </xsl:text>
  <xsl:for-each select="number[position() &lt;= $numCols]">
    <xsl:text>--------  </xsl:text>
  </xsl:for-each>
    <xsl:text>&#xa;</xsl:text>
  <xsl:for-each select="number[position() mod $numCols = 1]">
    <xsl:call-template name="pad">
      <xsl:with-param name="string">
        <xsl:number value="position()" format="i"/>
      </xsl:with-param>
      <xsl:with-param name="width" select="4"/>
    </xsl:call-template>|<xsl:text/> <!-- см. рецепт 7.1 -->
    <xsl:apply-templates
         select=". | following-sibling::number[position() &lt; $numCols]"
```

```
          mode="format"/>
      <xsl:text>&#xa;</xsl:text>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="number" name="format" mode="format">
    <xsl:param name="number" select="." />
    <xsl:call-template name="pad">
      <xsl:with-param name="string" select="format-number(.,'#,###.00')"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="pad">
    <xsl:param name="string"/>
    <xsl:param name="width" select="16"/>
    <xsl:call-template name="str:dup">
      <xsl:with-param name="input" select="' '"/>
      <xsl:with-param name="count" select="$width - string-length($string)"/>
    </xsl:call-template>
    <xsl:value-of select="$string"/>
  </xsl:template>

  </xsl:stylesheet>
```

Вывод:

```
           I             II            III
         -----------   -----------   -----------
      i|       10.00          3.50          4.44
     ii|       77.78         -8.00          1.00
    iii|      444.00          1.12          7.77
     iv|        3.14         10.00          9.00
      v|        8.00          7.00        666.00
     vi|    5,555.00 -4,444,444.00         22.33
    vii|       18.00         36.54         43.00
   viii|   99,999.00    999,999.00  9,999,999.00
     ix|       32.00         64.00        -64.00
```

Нумерация колонок, как в электронной таблице

В электронных таблицах колонки обозначаются буквами A, B, C, ... ZZ, и мы можем сделать то же самое с помощью элемента `xsl:number` (пример 3.7).

Пример 3.7. Нумерация колонок, как в электронной таблице

```
<xsl:template match="numbers">
  <xsl:for-each select="number[position() &lt;= $numCols]">
  <xsl:text>          </xsl:text>
```

```
    <xsl:number value="position()" format="A"/><xsl:text>   </xsl:text>
      <xsl:text>     </xsl:text>
    </xsl:for-each>
      <xsl:text>&#xa;</xsl:text>
    <xsl:for-each select="number[position() &lt;= $numCols]">
      <xsl:text>  -------- </xsl:text>
    </xsl:for-each>
      <xsl:text>&#xa;</xsl:text>
    <xsl:for-each select="number[position() mod $numCols = 1]">
      <xsl:value-of select="position()"/><xsl:text>|</xsl:text>
      <xsl:apply-templates
           select=". | following-sibling::number[position() &lt; $numCols]"
           mode="format"/>
      <xsl:text>&#xa;</xsl:text>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="number" name="format" mode="format">
    <xsl:param name="number" select="." />
    <xsl:call-template name="pad">
      <xsl:with-param name="string" select="format-number(.,'#,###.00')"/>
    </xsl:call-template>
    <xsl:text>   </xsl:text>
  </xsl:template>
```

Вывод:

```
         A                   B                  C
  -----------------  -----------------  -----------------
1|          10.0000             3.5000             4.4000
2|          77.7777            -8.0000             1.0000
3|         444.0000             1.1234             7.7700
4|           3.1416            10.0000             9.0000
5|           8.0000             7.0000           666.0000
6|        5555.0000      -4444444.0000            22.3300
7|          18.0000            36.5400            43.0000
8|       99999.0000        999999.0000       9999999.0000
9|          32.0000            64.0000           -64.0001
```

Запись чисел арабскими цифрами

Записать числа цифрами, используемыми в других языках, можно, задав в качестве атрибута `zero-digit` в элементе `xsl:decimal-format` обозначение нуля в соответствующем языке. В примере ниже мы взяли символ Unicode с кодом 0×660 (нуль в арабской нотации).

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings">

<xsl:output method="text" encoding="UTF-8"/>

<xsl:include href="../strings/str.dup.xslt"/>

<!-- Здесь говорится, что нуль находится в кодовой позиции 0x660,
следовательно, 1 занимает кодовую позицию 0x661 и т.д. -->
<xsl:decimal-format name="Arabic" zero-digit="&#x660;"/>

<xsl:template match="numbers">
  <xsl:for-each select="number">
      <xsl:call-template name="pad">
        <xsl:with-param name="string" select="format-
number(.,'#,###.00')"/>
      </xsl:call-template> = <xsl:text/>
      <xsl:value-of select="format-
number(.,'#,###.&#x660;&#x660;','Arabic')"/>
     <xsl:text>&#xa;</xsl:text>
  </xsl:for-each>
</xsl:template>

<xsl:template name="pad">
  <xsl:param name="string"/>
  <xsl:param name="width" select="16"/>
  <xsl:value-of select="$string"/>
  <xsl:call-template name="str:dup">
    <xsl:with-param name="input" select="''"/>
    <xsl:with-param name="count" select="$width - string-
length($string)"/>
  </xsl:call-template>
</xsl:template>

</xsl:stylesheet>
```

Этот код выводит следующий результат:

3.2. Округление чисел с заданной точностью

Задача

Требуется округлить число до заданного числа десятичных знаков после запятой. Однако функции XSLT `round`, `ceiling` и `floor` всегда возвращают целое число.

```
10.00 = ١٠.٠٠
3.50 = ٣.٥٠
4.44 = ٤.٤٤
77.78 = ٧٧.٧٨
-8.80 = −٨.٠٠
1.00 = ١.٠٠
444.00 = ٤٤٤.٠٠
1.12 = ١.١٢
7.77 = ٧.٧٧
3.14 = ٣.١٤
10.00 = ١٠.٠٠
9.00 = ٩.٠٠
8.00 = ٨.٠٠
7.00 = ٧.٠٠
666.00 = ٦٦٦.٠٠
5555.00 = ٥,٥٥٥.٠٠
-4444444.00 = −٤,٤٤٤,٤٤٤.٠٠
22.33 = ٢٢.٣٣
18.00 = ١٨.٠٠
36.54 = ٣٦.٥٤
43.00 = ٤٣.٠٠
99999.00 = ٩٩,٩٩٩.٠٠
999999.00 = ٩٩٩,٩٩٩.٠٠
9999999.00 = ٩,٩٩٩,٩٩٩.٠٠
32.00 = ٣٢.٠٠
64.00 = ٦٤.٠٠
-64.00 = −٦٤.٠٠
```

Решение

XSLT 1.0

Умножим на 10 в степени n, где n - требуемое число десятичных знаков, затем округлим и разделим на тот же коэффициент. Предположим, что `$pi = = 3/1415926535897932`. Тогда вычисление выражения

```
<xsl:value-of select="round($pi * 10000) div 10000"/>
```

дает 3.1416. Аналогично:

```
<xsl:value-of select="ceiling($pi * 10000) div 10000"/>
```

дает 3.1416, а:

```
<xsl:value-of select="floor($pi * 10000) div 10000"/>
```

дает 3.1415.

Округлить до заданного числа знаков можно также с помощью функции `format-number()`:

```
<xsl:value-of select="format-number($pi, '#.####')"/>
```

Получится 3.1416. Это решение будет работать, даже если в целой части всего одна значащая цифра, поскольку функция `format-number` никогда не интерпретирует спецификацию формата как указание удалять значащие цифры из целой части:

```
<xsl:value-of select="format-number($pi * 100, '#.####')"/>
```

В результате получаем 314.1593.

С помощью функции `format-number` можно получить эффект отбрасывания вместо округления, если задать в форматной строке на одну цифру больше, чем необходимо, а затем отбросить последний символ:

```
<xsl:variable name="pi-to-5-sig" select="format-number($pi, '#.#####')"/>
<xsl:value-of select="substring($pi-to-5-sig,1,
     string-length($pi-to-5-sig) - 1"/>
```

Результат будет равен 3.1415.

XSLT 2.0

В большинстве приложений задачу может решить появившаяся в XPath 2.0 функция `round-half-to-even()`. *Half to even* (округление половины до четного) означает, что в случае, когда округляемая величина оказывается точно посередине между большим и меньшим значением, округление производится в том направлении, где получится четный результат. Так, `round-half-to-even(1.115,2) eq 1.12`, и `round-half-to-even(1.125,2) eq 1.12`! В первом случае мы округляем с избытком, поскольку 2 – четное число, а во втором – с недостатком, так как 3 – число нечетное. Теоретически такой метод обосновывается тем, что если имеется много чисел, то округление с избытком должно происходить примерно так же часто, как с недостатком, поэтому ошибки округления будут взаимно уничтожаться. Вы, конечно, догадались, что второй аргумент функции `round-half-to-even()` – число десятичных знаков после округления.

Если в вашем приложении требуется, чтобы число, оканчивающееся на 5, всегда округлялось с избытком, то можете воспользоваться техникой, описанной в рецепте для XSLT 1.0.

Обсуждение

Метод «умножение, округление, деление» хорошо работает, если все промежуточные результаты не выходят за пределы представимости числа с плавающей точкой в формате IEEE. Если вы попробуете сохранить слишком много знаков после запятой, то результат будет искажен вследствие правил работы с числами с плавающей точкой. Например, попытка получить 16 десятичных знаков числа π даст всего 15:

```
<xsl:value-of select="round($pi * 10000000000000000) div 10000000000000000"/>
```

В результате получим 3.141592653589793, а не 3.1415926535897932.

Альтернативное решение – обращаться с числом, как со строкой, а затем обрезать лишние цифры:

```
<xsl:value-of select="concat(substring-before($pi,'.'),
           '.',
           substring(substring-after($pi,'.'),1,4))"/>
```

дает 3.1415.

Этот прием позволяет добиться того же эффекта, что `ceiling` или `round`, но ценой дополнительной сложности:

```
<xsl:variable name="whole" select="substring-before($pi,'.')"/>
<xsl:variable name="frac" select="substring-after($pi,'.')"/>
<xsl:value-of select="concat($whole,
            '.',
            substring($frac,1,3)),
     round(substring($frac,4,2) div 10))"/>
```

В результате получаем 3.1416.

3.3. Преобразование римских числительных в числа

Задача

Требуется преобразовать число, записанное римскими цифрами, в обычное число.

Решение

Римская система счисления не является позиционной. Число в ней записывается путем прибавления или вычитания фиксированного количества римских цифр. Если следующий символ имеет меньшее или такое же значение, то производится добавление, иначе вычитание.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://www.ora.com/XSLTCookbook/math" id="math:math.roman">

<ckbk:romans>
    <ckbk:roman value="1">i</ckbk:roman>
    <ckbk:roman value="1">I</ckbk:roman>
    <ckbk:roman value="5">v</ckbk:roman>
    <ckbk:roman value="5">V</ckbk:roman>
    <ckbk:roman value="10">x</ckbk:roman>
    <ckbk:roman value="10">X</ckbk:roman>
    <ckbk:roman value="50">l</ckbk:roman>
```

```
    <ckbk:roman value="50">L</ckbk:roman>
    <ckbk:roman value="100">c</ckbk:roman>
    <ckbk:roman value="100">C</ckbk:roman>
    <ckbk:roman value="500">d</ckbk:roman>
    <ckbk:roman value="500">D</ckbk:roman>
    <ckbk:roman value="1000">m</ckbk:roman>
    <ckbk:roman value="1000">M</ckbk:roman>
</ckbk:romans>

<xsl:variable name="ckbk:roman-nums" select="document('')/*/*/ckbk:roman"/>

<xsl:template name="ckbk:roman-to-number">
    <xsl:param name="roman"/>

    <xsl:variable name="valid-roman-chars">
           <xsl:value-of select="document('')/*/ckbk:romans"/>
    </xsl:variable>

    <xsl:choose>
        <!-- возвращает true, если в строке есть не-римские цифры -->
        <xsl:when test="translate($roman,$valid-roman-chars,'')">
           NaN</xsl:when>
        <xsl:otherwise>
           <xsl:call-template name="ckbk:roman-to-number-impl">
           <xsl:with-param name="roman" select="$roman"/>
          </xsl:call-template>
    </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:roman-to-number-impl">
  <xsl:param name="roman"/>
  <xsl:param name="value" select="0"/>

  <xsl:variable name="len" select="string-length($roman)"/>

  <xsl:choose>
    <xsl:when test="not($len)">
      <xsl:value-of select="$value"/>
    </xsl:when>
    <xsl:when test="$len = 1">
      <xsl:value-of select="$value + $ckbk:roman-nums[. = $roman]/@value"/>
    </xsl:when>
    <xsl:otherwise>
```

```
      <xsl:variable name="roman-num"
            select="$ckbk:roman-nums[. = substring($roman, 1, 1)]"/>
      <xsl:choose>
         <xsl:when test="$roman-num/following-sibling::ckbk:roman =
           substring($roman, 2, 1)">
            <xsl:call-template name="ckbk:roman-to-number-impl">
            <xsl:with-param name="roman"
                  select="substring($roman,2,$len - 1)"/>
            <xsl:with-param name="value"
                  select="$value - $roman-num/@value"/>
         </xsl:call-template>
      </xsl:when>
    <xsl:otherwise>
       <xsl:call-template name="ckbk:roman-to-number-impl">
         <xsl:with-param name="roman" select="substring($roman,2,$len - 1)"/>
         <xsl:with-param name="value" select="$value + $roman-num/@value"/>
       </xsl:call-template>
      </xsl:otherwise>
     </xsl:choose>
    </xsl:otherwise>
   </xsl:choose>

</xsl:template>
</xsl:stylesheet>
```

Обсуждение

Элемент `xsl:number` дает удобный способ преобразовать число в римскую запись. Однако обратное преобразование приходится выполнять самостоятельно. Показанный выше рекурсивный шаблон решает эту задачу прямолинейно и очень похож на приведенный в книге Jeni Tennison *XSLT and XPath on the Edge* (M&T Books, 2001).

Имеются два мелких подводных камня, но в большинстве случаев они не приводят к неприятностям. Первый заключается в том, что это решение не работает, когда число записано римскими цифрами в разных регистрах (например, `IiI`). Такие аномалии вряд ли встретятся в нормальном источнике данных, тем не менее приведенный код и не отбросит входные данных как некорректные, и правильного результата не даст. Если добавить преобразование к единому регистру, то некорректные входные данные можно будет либо отвергнуть, либо нормально обработать.

Вторая неприятность связана с тем фактом, что не существует стандартного представления в римской записи чисел, больших 1000. Процессоры Saxon и Xalan сцепляют в этом случае буквы M, но другие могут поступать иначе.

Если по какой-то причине вам не нравится хранить данных о римских числительных в таблице стилей, то можете воспользоваться следующим выражением на языке XPath 1.0 для декодирования:

```
<xsl:variable name="roman-value"
    select="($c = 'i' or $c = 'I') * 1 +
                  ($c = 'v' or $c = 'V') * 5 +
                  ($c = 'x' or $c = 'X') * 10 +
                  ($c = 'l' or $c = 'L') * 50 +
                  ($c = 'c' or $c = 'C') * 100 +
                  ($c = 'd' or $c = 'D') * 500 +
                  ($c = 'm' or $c = 'M') * 1000)"/>
```

В XSLT 2.0 можно воспользоваться вложенным выражением if-the-else или организовать справочную таблицу из последовательности:

```
<xsl:variable name="roman-value" select="(1,5,10,50,100,500,1000)
    [index-of(('I', 'V', 'X', 'L', 'C', 'D', 'M'), upper-case($c))]"/>
```

3.4. Преобразование из одной системы счисления в другую

Задача

Требуется преобразовать строку, представляющую число в системе с одним основанием, в число, записанное в системе с другим основанием.

Решение

В следующем примере приведено общее решение для преобразования из системы с любым основанием от 2 до 36 в систему с другим основанием из того же диапазона. Здесь используются две глобальные переменные для кодирования всех символов в системе с основанием 36 в виде смещений от начала строки, одной для кодирования заглавных букв и другой – для прописных.

```
<xsl:variable name="ckbk:base-lower"
  select="'0123456789abcdefghijklmnopqrstuvwxyz'"/>

<xsl:variable name="ckbk:base-upper"
  select="'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'"/>

<xsl:template name="ckbk:convert-base">
   <xsl:param name="number"/>
   <xsl:param name="from-base"/>
   <xsl:param name="to-base"/>

   <xsl:variable name="number-base10">
     <xsl:call-template name="ckbk:convert-to-base-10">
        <xsl:with-param name="number" select="$number"/>
        <xsl:with-param name="from-base" select="$from-base"/>
```

```
      </xsl:call-template>
    </xsl:variable>
    <xsl:call-template name="ckbk:convert-from-base-10">
       <xsl:with-param name="number" select="$number-base10"/>
       <xsl:with-param name="to-base" select="$to-base"/>
    </xsl:call-template>
</xsl:template>
```

В этом решении общая задача сведена к двум подзадачам: преобразование в систему с основанием 10 и из нее. Преобразования в десятичной системе проще, потому что именно в ней представляются числа в языке XPath.

Шаблон `ckbk:convert-to-base-10` нормализует входное число, приводя его к нижнему регистру. Иными словами, шестнадцатеричные числа `ffff` и `FFFF` – одно и то же, что соответствует общепринятому соглашению. Две выполняемые проверки призваны гарантировать, что основание принадлежит допустимому диапазону, а число не содержит символов, не разрешенных при таком основании. Обрабатывается также тривиальный случай преобразования из десятичной системы в десятичную:

```
<xsl:template name="ckbk:convert-to-base-10">
  <xsl:param name="number"/>
  <xsl:param name="from-base"/>
  <xsl:variable name="num"
          select="translate($number,$ckbk:base-upper, $ckbk:base-lower)"/>
  <xsl:variable name="valid-in-chars"
          select="substring($ckbk:base-lower,1,$from-base)"/>

  <xsl:choose>
    <xsl:when test="$from-base &lt; 2 or $from-base > 36">NaN</xsl:when>
    <xsl:when test="not($num) or translate($num,$valid-in-chars,'')">
         NaN</xsl:when>
    <xsl:when test="$from-base = 10">
      <xsl:value-of select="$number"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="ckbk:convert-to-base-10-impl">
         <xsl:with-param name="number" select="$num"/>
         <xsl:with-param name="from-base" select="$from-base"/>
         <xsl:with-param name="from-chars" select="$valid-in-chars"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

После того как все проверки выполнены, собственно преобразование делегируется другому рекурсивному шаблону. Он находит десятичное значение каждого

символа как смещение от начала строки, полученной от вызывающего шаблона. Рекурсивные вызовы умножают текущий результат на основание и прибавляют к произведению значение первого символа, пока не останется строка длины 1.

```
<xsl:template name="ckbk:convert-to-base-10-impl">
  <xsl:param name="number"/>
  <xsl:param name="from-base"/>
  <xsl:param name="from-chars"/>

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

  <xsl:variable name="value"
      select="string-length(substring-before($from-chars,
                            substring($number,1,1)))"/>

  <xsl:variable name="total" select="$result * $from-base + $value"/>

  <xsl:choose>
    <xsl:when test="string-length($number) = 1">
      <xsl:value-of select="$total"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="ckbk:convert-to-base-10-impl">
         <xsl:with-param name="number" select="substring($number,2)"/>
         <xsl:with-param name="from-base" select="$from-base"/>
         <xsl:with-param name="from-chars" select="$from-chars"/>
         <xsl:with-param name="result" select="$total"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Для решения второй подзадачи нужно уметь преобразовывать из десятичной системы в систему с любым другим основанием. И снова проверка ошибок отделяется от самого преобразования.

```
<xsl:template name="ckbk:convert-from-base-10">
  <xsl:param name="number"/>
  <xsl:param name="to-base"/>

  <xsl:choose>
    <xsl:when test="$to-base &lt; 2 or $to-base > 36">NaN</xsl:when>
    <xsl:when test="number($number) != number($number)">NaN</xsl:when>
    <xsl:when test="$to-base = 10">
      <xsl:value-of select="$number"/>
    </xsl:when>
```

```
    <xsl:otherwise>
      <xsl:call-template name="ckbk:convert-from-base-10-impl">
         <xsl:with-param name="number" select="$number"/>
         <xsl:with-param name="to-base" select="$to-base"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Для преобразования необходимо найти символ, отстоящий от начала строки `ckbk:base-lower` на величину, равную остатку от его деления на `$to-base`. Рекурсия производится по оставшейся целой части после приписывания получившейся цифры в начало текущего результата. Завершается рекурсия, когда число окажется меньше основания.

```
<xsl:template name="ckbk:convert-from-base-10-impl">
  <xsl:param name="number"/>
  <xsl:param name="to-base"/>
  <xsl:param name="result"/>

  <xsl:variable name="to-base-digit"
    select="substring($ckbk:base-lower,$number mod $to-base + 1,1)"/>

  <xsl:choose>
    <xsl:when test="$number >= $to-base">
     <xsl:call-template name="ckbk:convert-from-base-10-impl">
       <xsl:with-param name="number" select="floor($number div $to-base)"/>
       <xsl:with-param name="to-base" select="$to-base"/>
       <xsl:with-param name="result" select="concat($to-base-digit,$result)"/>
     </xsl:call-template>
    </xsl:when>
   <xsl:otherwise>
      <xsl:value-of select="concat($to-base-digit,$result)"/>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Обсуждение

Преобразование из одной системы счисления в другую – распространенная в программировании задача, поэтому большинство разработчиков знает, как это делается. Во многих языках имеются готовые средства для решения этой задачи, но в XSLT их нет. Из-за того, что в XPath 1.0 и XSLT 1.0 нет функций для получения целочисленного значения символа Unicode, код оказывается громоздким. В XPath 2.0 можно воспользоваться функциями `string-to-codepoints` и `codepoints-to-string`. Для обращения со строками как со справочными таблицами приходится прибегать к хитрым

трюкам. Хотя такие манипуляции неэффективны, для большинства задач, в которых встречаются преобразования системы счисления, с этим можно смириться.

В приведенном выше коде предполагается, что при основании большем 10 применяется стандартное соглашение об обозначении цифр, больших 9, последовательными буквами. Если вы сталкиваетесь с нестандартным кодированием, то нужно будет соответственно изменить строки, играющие роль справочников. В принципе можно обобщить этот код на случай основания больше 36, добавив символы, которыми будут кодироваться дополнительные цифры.

3.5. Реализация стандартных математических функций

Задача

Требуется выйти за пределы математики для пятого класса, но в XSLT необходимых средств нет.

Решение

XSLT 1.0

Ниже предлагаются реализации на языке XSLT 1.0 функций для вычисления абсолютного значения, квадратного корня, логарифмов, степенной функции и факториала.

Абсолютное значение ckbk:abs(x)

Вот очевидный, но многословный способ получить абсолютное значение числа:

```
<xsl:template name="ckbk:abs">
     <xsl:param name="x"/>

     <xsl:choose>
          <xsl:when test="$x &lt; 0">
               <xsl:value-of select="$x * -1"/>
          </xsl:when>
          <xsl:otherwise>
               <xsl:value-of select="$x"/>
          </xsl:otherwise>
     </xsl:choose>

</xsl:template>
```

Короткий, но с первого взгляда не понятный способ основан на том факте, что значение `true` преобразуется в число 1, а `false` – в 0:

```
<xsl:template name="ckbk:abs">
    <xsl:param name="x"/>
```

```
    <xsl:value-of select="(1 - 2*($x &lt; 0)) * $x"/>
</xsl:template>
```

Этот вариант мне нравится больше из-за своей краткости. Можно также реализовать эту функцию в виде расширения (см. главу 12).

Квадратный корень ckbk:sqrt(x)

Нэйт Остин (Nate Austin) предложил для проекта EXSLT реализацию квадратного корня на чистом XSLT с помощью метода Ньютона.

```
<xsl:template name="ckbk:sqrt">
    <!-- Число, из которого извлекается квадратный корень. -->
    <xsl:param name="number" select="0"/>
    <!-- Текущее "приближение".  Внутренняя переменная. -->
    <xsl:param name="try" select="1"/>
    <!-- Номер текущей итерации. Сравнивается с maxiter для ограничения
    количества повторений цикла. -->
    <xsl:param name="iter" select="1"/>
    <!-- Этот параметр предотвращает бесконечный цикл. -->
    <xsl:param name="maxiter" select="20"/>
    <!-- Этот шаблон написал Нэйт Остин, применив метод Ньютона для
    извлечения корня. ->
    <xsl:choose>
       <xsl:when test="$try * $try = $number or $iter > $maxiter">
          <xsl:value-of select="$try"/>
       </xsl:when>
       <xsl:otherwise>
          <xsl:call-template name="ckbk:sqrt">
           <xsl:with-param name="number" select="$number"/>
           <xsl:with-param name="try" select="$try -
                           (($try * $try - $number) div (2 * $try))"/>
           <xsl:with-param name="iter" select="$iter + 1"/>
           <xsl:with-param name="maxiter" select="$maxiter"/>
          </xsl:call-template>
       </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Изменение начального значения параметра `try` может заметно повысить производительность:

```
<xsl:template name="math:sqrt">
    <!-- Число, из которого извлекается квадратный корень. -->
    <xsl:param name="number" select="0"/>
    <!-- Текущее "приближение". Внутренняя переменная. -->
    <xsl:param name="try" select="($number &lt; 100) +
```

```
            ($number >= 100 and $number &lt; 1000) * 10 +
            ($number >= 1000 and $number &lt; 10000) * 31 +
            ($number >= 10000) * 100"/>
    <!-- Больше ничего не меняется. -->
</xsl:template>
```

Этот нехитрый трюк (в котором снова применяется преобразование булевских значений в числовые) позволяет подобрать величину `try` так, чтобы она лучше аппроксимировала квадратный корень на самой первой итерации. Вычислив в тестовой программе корни из всех чисел от 1 до 10000, я получил выигрыш 10%. Но гораздо важнее, что за счет этой модификации средняя погрешность снизилась с 1×10^{-5} до 6×10^{-13}. Это означает, что для получения той же точности можно сократить число итераций и, значит, еще увеличить быстродействие. Например, уменьшив число итераций с 10 до 6, я получил ту же погрешность 1×10^{-5}, повысив производительность вдвое. Если вам нужно извлекать квадратные корни из чисел, гораздо больших 10000, то лучше задать по крайней мере 10 итераций и добавить дополнительные диапазоны в инициализацию `try`.

Логарифмы: ckbk:log10(number), ckbk:log(number) и ckbk:logN(x,base)

Если имеющийся у вас процессор XSLT поддерживает функцию `exsl:log()` (натуральный логарифм), то реализовать логарифм по любому другому основанию просто. На псевдокоде это выглядит так:

```
<!-- Основное правило логарифмов -->
ckbk:logN(x,base) = ckbk:logN(x,diffBase) div ckbk:logN(base,diffBase)
```

К сожалению, ни один из процессоров, перечисленных на сайте EXSLT.org, пока не поддерживает функцию `exsl:log`. Следующий вариант – реализовать расширение на языке Java, воспользовавшись классом `java.lang.Math.log` или функцией `Math.log` в JavaScript. Наконец, можно не прибегать к расширениям вовсе, а написать log10 на чистом XSLT с приемлемой для большинства приложений точностью и производительностью. Имея реализацию `log10()`, можно вычислять произвольные логарифмы, применяя основное правило.

```
<xsl:template name="ckbk:log10">
    <xsl:param name="number" select="1"/>

    <xsl:param name="n" select="0"/>  <!-- служебная переменная для
                                хранения целой части результата -->

    <xsl:choose>
       <!-- Для нуля и отрицательных чисел логарифм не определен -->
       <xsl:when test="$number &lt;= 0">
       <xsl:value-of select="NaN"/>
    </xsl:when>
```

```
    <xsl:when test="$number &lt; 1"> <!-- Логарифм числа, меньшего 1,
                                отрицателен -->
       <xsl:call-template name="ckbk:log10">
          <xsl:with-param name="number" select="$number * 10"/>
          <xsl:with-param name="n" select="$n - 1"/>
       </xsl:call-template>
    </xsl:when>
    <xsl:when test="$number > 10"> <!-- Если число больше 10,
                         его логарифм больше 1 -->
       <xsl:call-template name="ckbk:log10">
           <xsl:with-param name="number" select="$number div 10"/>
           <xsl:with-param name="n" select="$n + 1"/>
       </xsl:call-template>
    </xsl:when>
    <xsl:when test="$number = 10">
           <xsl:value-of select="$n + 1"/>
    </xsl:when>
    <xsl:otherwise>    <!-- Нам нужно лишь уметь вычислить логарифмы
                чисел в диапазоне [1,10) -->
       <xsl:call-template name="ckbk:log10-util">
           <xsl:with-param name="number" select="$number"/>
           <xsl:with-param name="n" select="$n"/>
       </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Вычисляет натуральный логарифм числа -->
<xsl:template name="ckbk:log">
    <xsl:param name="number" select="1"/>

    <xsl:variable name="log10-e" select="0.4342944819"/>
    <xsl:variable name="log10">
        <xsl:call-template name="ckbk:log10">
           <xsl:with-param name="number" select="$number"/>
        </xsl:call-template>
    </xsl:variable>
    <xsl:value-of select="$log10 div $log10-e"/>
</xsl:template>

<!-- Вычисляет логарифм числа по основанию b -->
<xsl:template name="ckbk:log-b">
    <xsl:param name="number" select="1"/>
    <xsl:param name="base" select="2"/>
```

```
    <xsl:variable name="log10-base">
       <xsl:call-template name="ckbk:log10">
         <xsl:with-param name="number" select="$base"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:variable name="log10">
    <xsl:call-template name="ckbk:log10">
      <xsl:with-param name="number" select="$number"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:value-of select="$log10 div $log10-base"/>
</xsl:template>

<!-- Вычисляет log10 для чисел в диапазоне [1,10) + n-->
<xsl:template name="ckbk:log10-util">
    <xsl:param name="number"/>

    <xsl:param name="n"/>
    <xsl:param name="frac" select="0"/>
         <!-- служебная переменная для хранения дробной части -->
    <xsl:param name="k" select="0"/>          <!-- счетчик итераций -->
    <xsl:param name="divisor" select="2"/>
         <!-- последовательные степени 2 для построения дробной части -->
    <xsl:param name="maxiter" select="38"/>
         <!-- Число итераций. 38 более чем достаточно для получения
         не менее 10 точных цифр -->

    <xsl:variable name="x" select="$number * $number"/>

    <xsl:choose>
       <xsl:when test="$k >= $maxiter">
          <!-- Округлить до 10 значащих десятичных цифр -->
          <xsl:value-of select="$n + round($frac * 10000000000) div
              10000000000"/>
       </xsl:when>
       <xsl:when test="$x &lt; 10">
          <xsl:call-template name="ckbk:log10-util">
           <xsl:with-param name="number" select="$x"/>
           <xsl:with-param name="n" select="$n"/>
           <xsl:with-param name="k" select="$k + 1"/>
           <xsl:with-param name="divisor" select="$divisor * 2"/>
           <xsl:with-param name="frac" select="$frac"/>
           <xsl:with-param name="maxiter" select="$maxiter"/>
```

```
            </xsl:call-template>
          </xsl:when>
          <xsl:otherwise>
            <xsl:call-template name="ckbk:log10-util">
            <xsl:with-param name="number" select="$x div 10"/>
            <xsl:with-param name="n" select="$n"/>
            <xsl:with-param name="k" select="$k + 1"/>
            <xsl:with-param name="divisor" select="$divisor * 2"/>
            <xsl:with-param name="frac" select="$frac + (1 div $divisor)"/>
            <xsl:with-param name="maxiter" select="$maxiter"/>
            </xsl:call-template>
          </xsl:otherwise>
      </xsl:choose>
  </xsl:template>
```

Основная задача шаблона `ckbk:log10` – свести задачу вычисления `log10(x)` к более простой задаче вычисления `log10(x : 1 <= x < 10)`. Для этого заметим, что `log10(x : x > 10) = log10(x div 10) + 1` и `log10(x : x < 1) = = log10(x * 10) - 1`. Кроме того, контролируются входные данные, так как для нуля и отрицательных чисел логарифм не определен.

Вспомогательный шаблон `ckbk:log10-util` делает всю сложную работу. Это основанная на хвостовой рекурсии реализация итеративного алгоритма, описанного в книге Кнута[1]. Чтобы организовать хвостовую рекурсию и существенно упростить реализацию, мы ввели несколько служебных параметров:

`n`

Целая часть результата, передаваемая в шаблон `ckbk:log10`. Без этого параметра можно было бы обойтись, так как передаваемую величину можно хранить, пока `ckbk:log10-util` занимается своей работой. Однако он позволяет избежать необходимости сохранять результат `ckbk:log10-util` в переменной.

`frac`

Дробная часть результата. Именно ее мы и ищем.

`k`

Счетчик итераций, который увеличивается на 1 при каждом рекурсивном вызове. Рекурсия прекращается, как только `$k > $maxiter`.

`divisor`

Число, которому присваивается следующая по порядку степень 2 при каждом рекурсивном вызове (т.е. 2, 4, 8, 16, ...). Величина `1 div $divisor` прибавляется к `frac` по мере аппроксимации логарифма.

`maxiter`

[1] Стоящая за ним математическая теория выходит за рамки данной книги. См. Д. Кнут «Искусство программирования», т. 1, издательство «Вильямс», 2007.

Количество итераций для вычисления frac. Чем больше `maxiter`, тем выше точность результата (в границах, определяемых представлением числа с плавающей точкой в формате IEEE). Этот параметр можно было бы и опустить, но он позволяет вызывающей программе задать требуемое число итераций и тем самым установить компромисс между быстродействием и точностью.

Степенная функция: ckbk:power(base,power)

В настоящий момент на сайте EXSLT.org не упомянуты процессоры, поддерживающие функцию `ckbk:power()`. Однако она определена в проекте EXSLT и может быть легко реализована на чистом XSLT. Джени Теннисон предлагает такой шаблон:

```
<xsl:template name="ckbk:power">
    <xsl:param name="base"/>
    <xsl:param name="power"/>
    <xsl:choose>
         <xsl:when test="$power = 0">
           <xsl:value-of select="$result"/>
         </xsl:when>
           <xsl:otherwise>
           <xsl:variable name="temp">
              <xsl:call-template name="ckbk:power">
              <xsl:with-param name="base" select="$base"/>
              <xsl:with-param name="power" select="$power - 1"/>
              </xsl:call-template>
              </xsl:variable>
           <xsl:value-of select="$base * $temp"/>
         </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Для большинства приложений этот код вполне приемлем. Однако рекурсия в нем не хвостовая и алгоритмически он не самый эффективный. Следующая реализация лишена первого недостатка, а число операций умножения в ней уменьшилось с `O($power)` до `O(log2($power)`. Кроме того, добавлена обработка ошибок, предотвращающая бесконечную рекурсию в случае, когда `$power` равно `NaN`.

```
<xsl:template name="ckbk:power">
    <xsl:param name="base"/>
    <xsl:param name="power"/>
    <xsl:param name="result" select="1"/>
    <xsl:choose>
       <xsl:when test="number($base) != $base or number($power) != $power">
           <xsl:value-of select="'NaN'"/>
       </xsl:when>
```

```
        <xsl:when test="$power = 0">
           <xsl:value-of select="$result"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="ckbk:power">
            <xsl:with-param name="base" select="$base * $base"/>
             <xsl:with-param name="power" select="floor($power div 2)"/>
             <xsl:with-param name="result"
                   select="$result * $base * ($power mod 2) +
                             $result * not($power mod 2)"/>
             </xsl:call-template>
         </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Здесь мы ввели служебный параметр `$result`, в котором строится окончательный результат. Он и позволяет организовать хвостовую рекурсию. При каждом рекурсивном вызове основание возводится в квадрат, а показатель степени уменьшается вдвое. Поскольку мы пользуемся функцией `floor()`, `$result` достигнет 0 за `ceiling(log2($power))` рекурсивных вызовов. Это повышает производительность. Хитроумная часть заключается в вычислении `$result` на каждом шаге.

Проанализируем это выражение, обращая внимание на оба слагаемых. Выражение `$result * $base * ($power mod 2)` равно `$result * $base`, если `$power` нечетно и 0 в противном случае. Наоборот, $result * not($power mod 2) равно 0, когда `$power` четно и `$result` в противном случае. В XPath 2.0 это можно было бы записать как `if ($power % 2) then result *base else result`. А в XPath 1.0 приходится прибегать к трюкам. Так или иначе, шаблон вычисляет сумму $b^1 * base + b^2 * base^2 + b^3 * base^4 + b^4 * base^8$ ..., где b^i попеременно равно 0 или 1. Легко видеть, что суммирование может производиться до $base^{power}$ для произвольного целого числа power, если задавать в качестве b^i подходящие значения, что как раз и делает выражение `$power mod 2`. Если идея осталась непонятной, проработайте на бумаге несколько примеров, чтобы убедиться, что все работает правильно[1].

Показанная выше реализация степенной функции умеет вычислять только целые положительные степени. Однако, как известно любому студенту-математику, x^y является вещественным числом для всех вещественных x и y, а не только для натуральных показателей степени. Хорошо было бы иметь общую версию функции `power()`, вдруг пригодится. Ниже мы приводим соответствующий код.

[1] Хотя формальное доказательство правильности алгоритма – дело важное, для меня заниматься этим скучно и утомительно, поэтому ничего подобного вы в этой книге не найдете. Я предпочитаю полагаться на интуицию и тестирование. В главе 12 показано, как однородность языка XSLT, в котором данные и программы представляются в одной и той же форме, существенно упрощает тестирование. Настоящий программист-теоретик доказывал бы все логически, не нуждаясь в тестировании.

Чтобы не путать его с `power()`, шаблон называется `power-f()`, где f происходит от «floating point» (с плавающей точкой). Если хотите, можете назвать более общую версию `power()`, только внесите изменения в код ниже. Впрочем, иметь более специализированную версию в качестве отдельной функции тоже полезно.

```
<xsl:template name="ckbk:power-f">
    <xsl:param name="base"/>
    <xsl:param name="power"/>

    <xsl:choose>
       <xsl:when test="number($base) != $base or number($power) != $power">
          <xsl:value-of select="'NaN'"/>
       </xsl:when>
       <xsl:when test="$power &lt; 0">
          <xsl:variable name="result">
          <xsl:call-template name="ckbk:power-f">
           <xsl:with-param name="base" select="$base"/>
           <xsl:with-param name="power" select="-1 * $power"/>
          </xsl:call-template>
          </xsl:variable>
          <xsl:value-of select="1 div $result"/>
       </xsl:when>
       <xsl:otherwise>
           <xsl:variable name="powerN" select="floor($power)"/>
           <xsl:variable name="resultN">
           <xsl:call-template name="ckbk:power">
           <xsl:with-param name="base" select="$base"/>
           <xsl:with-param name="power" select="$powerN"/>
           </xsl:call-template>
           </xsl:variable>
           <xsl:choose>
             <xsl:when test="$power - $powerN">
              <xsl:variable name="resultF">
                <xsl:call-template name="ckbk:power-frac">
                <xsl:with-param name="base" select="$base"/>
              <xsl:with-param name="power" select="$power - $powerN"/>
             </xsl:call-template>
           </xsl:variable>
           <xsl:value-of select="$resultN * $resultF"/>
          </xsl:when>
          <xsl:otherwise>
              <xsl:value-of select="$resultN"/>
            </xsl:otherwise>
          </xsl:choose>
       </xsl:otherwise>
```

```
  </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:power-frac">
    <xsl:param name="base"/>
    <xsl:param name="power"/>

    <xsl:param name="n" select="1"/>
    <xsl:param name="ln_base">
       <xsl:call-template name="ckbk:log">
           <xsl:with-param name="number" select="$base"/>
       </xsl:call-template>
    </xsl:param>
    <xsl:param name="ln_base_n" select="$ln_base"/>
    <xsl:param name="power_n" select="$power"/>
    <xsl:param name="n_fact" select="$n"/>
    <xsl:param name="result" select="1"/>

     <xsl:choose>
       <xsl:when test="20 >= $n">
          <xsl:call-template name="ckbk:power-frac">
           <xsl:with-param name="base" select="$base"/>
           <xsl:with-param name="power" select="$power"/>
           <xsl:with-param name="n" select="$n + 1"/>
           <xsl:with-param name="ln_base" select="$ln_base "/>
           <xsl:with-param name="ln_base_n" select="$ln_base_n * $ln_base"/>
           <xsl:with-param name="power_n" select="$power_n * $power"/>
           <xsl:with-param name="n_fact" select="$n_fact * ($n+1)"/>
           <xsl:with-param name="result" select="$result +
                        ($power_n * $ln_base_n) div $n_fact"/>
          </xsl:call-template>
       </xsl:when>
    <xsl:otherwise>
       <xsl:value-of select="round($result * 1000000000) div 1000000000"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>3
```

Основные вычисления производятся не в шаблоне `ckbk:power-f`. Он лишь проверяет входные данные, а затем делегирует работу уже имеющемуся шаблону `ckbk:power` и новому `ckbk:power-frac`. В шаблоне `ckbk:power-f` используются два факта, касающихся степенной функции:

- $base^{-y} = 1 / base^{y}$, это позволяет легко возводить в отрицательную степень;
- $base^{(power1+power2)} = base^{power1} + base^{power2}$, это дает возможность повторно использовать точность и эффективность шаблона `ckbk:power` для целой части `$power` и получать хорошее приближение для дробной части.

Шаблон `ckbk:power-frac` представляет собой рекурсивную реализацию разложения в ряд Маклорена для x^y (см. рис. 3.1).

$$\text{Power} = [\text{base, power}] := 1{,}0 + \sum_{i=1}^{n} \frac{\text{Log}[\text{base}]^i \text{power}^i}{i!}$$

Рис. 3.1. Разложение степенной функции в ряд Маклорена

Один из способов реализации тригонометрических функций также основан на аналогичном разложении в ряд Маклорена.

$$\text{Sin}\,(x) = \sum_{i=0}^{\infty} \frac{-1^i x^{1+2i}}{(1+2i)!}$$

$$\text{Cos}\,(x) = \sum_{i=0}^{\infty} \frac{-1^i x^{2i}}{(2i)!}$$

Факториал

Как это ни странно, в проекте EXSLT не определена функция для вычисления факториала. Разумеется, реализовать ее несложно:

```
<xsl:template name="ckbk:fact">
    <xsl:param name="number" select="0"/>
    <xsl:param name="result" select="1"/>
    <xsl:choose>
       <xsl:when test="$number &lt; 0 or floor($number) != $number">
          <xsl:value-of select="'NaN'"/>
       </xsl:when>
       <xsl:when test="$number &lt; 2">
          <xsl:value-of select="$result"/>
       </xsl:when>
       <xsl:otherwise>
          <xsl:call-template name="ckbk:fact">
          <xsl:with-param name="number" select="$number - 1"/>
          <xsl:with-param name="result" select="$number * $result"/>
       </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Полезное обобщение факториала – это функция, которая вычисляет произведение всех чисел из заданного диапазона:

```
<xsl:template name="ckbk:prod-range">
    <xsl:param name="start" select="1"/>
    <xsl:param name="end" select="1"/>
    <xsl:param name="result" select="1"/>
    <xsl:choose>
       <xsl:when test="$start > $end">
          <xsl:value-of select="$result"/>
       </xsl:when>
        <xsl:otherwise>
           <xsl:call-template name="ckbk:prod-range">
           <xsl:with-param name="start" select="$start + 1"/>
           <xsl:with-param name="end" select="$end"/>
           <xsl:with-param name="result" select="$start * $result"/>
        </xsl:call-template>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

XSLT 2.0

В версии 2.0 функция `abs()` встроена. Прочие можно для пущего удобства реализовать как функции.

```
<!-- Степенная функция -->
<xsl:function name="ckbk:power" as="xs:double">
    <xsl:param name="base" as="xs:double"/>
    <xsl:param name="exp" as="xs:integer"/>
    <xsl:sequence
         select="if ($exp lt 0) then ckbk:power(1.0 div $base, -$exp)
                 else
                 if ($exp eq 0)
                  then 1e0 else $base * ckbk:power($base, $exp - 1)"/>
</xsl:function>

<!-- Квадратный корень -->
<xsl:function name="ckbk:sqrt" as="xs:double">
    <xsl:param name="number" as="xs:double"/>
    <xsl:variable name="try" select="if ($number lt 100.0) then 1.0
                                else if ($number gt 100.0 and $number lt
                                          1000.0) then 10.0
                                else if ($number gt 1000.0 and $number lt
                                          10000.0) then 31.0
                                else 100.00" as="xs:decimal"/>

    <xsl:sequence select="if ($number ge 0) then ckbk:sqrt($number,$try,1,20)
                          else 'NaN'"/>
```

```
</xsl:function>

<xsl:function name="ckbk:sqrt" as="xs:double">
    <xsl:param name="number" as="xs:double"/>
    <xsl:param name="try" as="xs:double"/>
    <xsl:param name="iter" as="xs:integer"/>
    <xsl:param name="maxiter" as="xs:integer"/>

    <xsl:variable name="result" select="$try * $try" as="xs:double"/>
    <xsl:sequence select="if ($result eq $number or $iter gt $maxiter)
                  then $try
                  else ckbk:sqrt($number, ($try - (($result - $number)
                               div (2 * $try))), $iter + 1, $maxiter)"/>
</xsl:function>

<!-- Факториал -->
<xsl:function name="ckbk:factorial" as="xs:decimal">
    <xsl:param name="n" as="xs:integer"/>
    <xsl:sequence select="if ($n eq 0) then 1 else $n *
ckbk:factorial($n - 1)"/>
</xsl:function>

<!-- Prod-range -->
<xsl:function name="ckbk:prod-range" as="xs:decimal">
    <xsl:param name="from" as="xs:integer"/>
    <xsl:param name="to" as="xs:integer"/>
    <xsl:sequence select="if ($from ge $to)
                          then $from
                          else $from * ckbk:prod-range($from + 1, $to)"/>
</xsl:function>

<!-- Log10 -->
<xsl:function name="ckbk:log10" as="xs:double">
    <xsl:param name="number" as="xs:double"/>
    <xsl:sequence select="if ($number le 0)
                           then 'NaN' else ckbk:log10($number,0)"/>
</xsl:function>

<xsl:function name="ckbk:log10" as="xs:double">
    <xsl:param name="number" as="xs:double"/>
    <xsl:param name="n" as="xs:double"/>
    <xsl:sequence select="if ($number le 1)
           then ckbk:log10($number * 10, $n - 1)
           else if($number gt 10)
           then ckbk:log10($number div 10, $n + 1)
```

```
            else if($number eq 10)
            then $n + 1
            else $n + ckbk:log10-util($number,0,0,2,38)"/>
</xsl:function>

<xsl:function name="ckbk:log10-util" as="xs:double">
    <xsl:param name="number" as="xs:double"/>
    <xsl:param name="frac" as="xs:double"/>
    <xsl:param name="iter" as="xs:integer"/>
    <xsl:param name="divisor" as="xs:double"/>
    <xsl:param name="maxiter" as="xs:integer"/>

    <xsl:variable name="x"select="$number * $number"/>

    <xsl:sequence select="if ($iter ge $maxiter)
                          then round-half-to-even($frac,10)
                          else if ($x lt 10)
                          then ckbk:log10-util($x,$frac,$iter + 1,
                                               $divisor * 2, $maxiter)
                          else ckbk:log10-util($x div 10,
                                               $frac + (1 div $divisor),
                          $iter + 1, $divisor * 2,
                          $maxiter)"/>
</xsl:function>
```

Обсуждение

Честно говоря, я думаю, что по меньшей мере в 80 процентах ваших приложений математика за пределами встроенных в XSLT возможностей никогда не понадобится. Ну а если в оставшихся 20 процентах случаев возникнет необходимость в функциях, отсутствующих в XSLT 1.0, то приведенные выше рецепты могут пригодиться.

Самый крупный недостаток реализаций на чистом XSLT 1.0 заключается в том, что шаблоны нельзя вызывать как настоящие функции из выражений на языке XPath. Поэтому математические операции записываются громоздко и оказываются не очень эффективными; ведь для запоминания вызовов шаблонов, представленных в виде результирующего фрагмента дерева, приходится создавать искусственные переменные. А процессор XSLT должен выполнять обратное преобразование этого фрагмента в число, когда впоследствии оно используется в вычислениях.

Еще одна проблема реализаций на XSLT состоит в том, что открытый интерфейс шаблонов замусорен служебными параметрами, которые лишь затемняют смысл функции. Эти параметры необходимы, чтобы организовать хвостовую рекурсию и предотвратить тем самым излишнюю работу. Например, в шаблоне `power-frac` логарифм основания вычисляется один раз и передается при каждом рекурсивном вызове. Если бы `ln_base` не был параметром, то логарифм пришлось

бы при каждом таком вызове вычислять заново, и производительность оказалось бы неприемлемо низкой.

XSLT 2.0 решает первую проблему, предоставляя полноценные функции. Поэтому в этой версии решения получаются более компактными и простыми. Но параметры команды `xsl:function` не могут принимать значения по умолчанию, так что этот недостаток приходится компенсировать, определяя перегруженные варианты функций с различной арностью. К сожалению, XSLT 2.0 не позволяет инкапсулировать закрытые параметры, которые необходимы при рекурсивных вызовах функций для служебных целей. Проектируя библиотеку функций, вы можете поместить внутренние реализации (те, в которых используются служебные параметры) в отдельное пространство имен, чтобы показать, что к таким функциям не следует обращаться напрямую.

Вторая проблема будет отчасти решена, если в будущих версиях XSLT появятся закрытые параметры, предназначенные только для рекурсивных вызовов.

3.6. Вычисление сумм и произведений

Задача

Требуется вычислить сумму или произведение чисел, содержащихся в наборе узлов.

Решение

XSLT 1.0

Абстрактная форма алгоритма суммирования для процессоров, оптимизирующих хвостовую рекурсию, выглядит следующим образом:

```
<xsl:template name="ckbk:sum">
  <!-- Инициализировать пустой набор узлов -->
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="result" select="'0'"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
       <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:otherwise>
        <!-- вызвать или применить шаблон, который определит значение
        узла в случае, когда сам узел не является суммируемым значением -->
      <xsl:variable name="value">
        <xsl:call-template name="some-function-of-a-node">
           <xsl:with-param name="node" select="$nodes[1]"/>
        </xsl:call-template>
    </xsl:variable>
       <!-- рекурсивно просуммировать по оставшейся части набора -->
```

```
      <xsl:call-template name="ckbk:sum">
         <xsl:with-param name="nodes" select="$nodes[position() != 1]"/>
         <xsl:with-param name="result" select="$result + $value"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Для обработки большого числа узлов при отсутствии поддержки хвостовой рекурсии есть два способа. Первый обычно называют *разделяй и властвуй*. Его идея в том, чтобы на каждом шаге рекурсии уменьшить объем работы по меньшей мере вдвое.

```
<xsl:template name="ckbk:sum-dvc">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="result" select="'0'"/>
  <xsl:param name="dvc-threshold" select="100"/>
  <xsl:choose>
    <xsl:when test="count($nodes) &lt;= $dvc-threshold">
       <xsl:call-template name="ckbk:sum">
          <xsl:with-param name="nodes" select="$nodes"/>
          <xsl:with-param name="result" select="$result"/>
       </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
       <xsl:variable name="half" select="floor(count($nodes) div 2)"/>
       <xsl:variable name="sum1">
          <xsl:call-template name="ckbk:sum-dvc">
            <xsl:with-param name="nodes"
                 select="$nodes[position() &lt;= $half]"/>
             <xsl:with-param name="result" select="$result"/>
              <xsl:with-param name="dvc-threshold" select="$dvc-threshold"/>
           </xsl:call-template>
        </xsl:variable>
        <xsl:call-template name="ckbk:sum-dvc">
           <xsl:with-param name="nodes" select="$nodes[position() > $half]"/>
           <xsl:with-param name="result" select="$sum1"/>
        <xsl:with-param name="dvc-threshold" select="$dvc-threshold"/>
       </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Второй способ называется *разбиением на части*. На первом этапе большая задача разбивается на части разумного размера, а на втором каждая часть обрабатывается рекурсивно.

```
<xsl:template name="ckbk:sum-batcher">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="result" select="'0'"/>
  <xsl:param name="batch-size" select="500"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
       <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:otherwise>
       <xsl:variable name="batch-sum">
          <xsl:call-template name="ckbk:sum">
           <xsl:with-param name="nodes"
                 select="$nodes[position() &lt; $batch-size]"/>
           <xsl:with-param name="result" select="$result"/>
          </xsl:call-template>
       </xsl:variable>
       <xsl:call-template name="ckbk:sum-batcher">
           <xsl:with-param name="nodes"
                select="$nodes[position() >= $batch-size]"/>
           <xsl:with-param name="result" select="$batch-sum"/>
           <xsl:with-param name="batch-size" select="$batch-size"/>
       </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Произведения вычисляются аналогично:

```
<xsl:template name="ckbk:product">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="result" select="1"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
       <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:otherwise>
        <!-- вызвать или применить шаблон, который определит значение
        узла в случае, когда сам узел не входит в число перемножаемых
        значений -->
       <xsl:variable name="value">
       <xsl:call-template name="some-function-of-a-node">
          <xsl:with-param name="node" select="$nodes[1]"/>
    </xsl:call-template>
    </xsl:variable>
       <xsl:call-template name="ckbk:product">
```

```
            <xsl:with-param name="nodes" select="$nodes[position() != 1]"/>
            <xsl:with-param name="result" select="$result * $value"/>
         </xsl:call-template>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:product-batcher">
   <xsl:param name="nodes" select="/.."/>
   <xsl:param name="result" select="1"/>
   <xsl:param name="batch-size" select="500"/>
   <xsl:choose>
      <xsl:when test="not($nodes)">
         <xsl:value-of select="$result"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:variable name="batch-product">
            <xsl:call-template name="ckbk:product">
               <xsl:with-param name="nodes"
                     select="$nodes[position() &lt; $batch-size]"/>
               <xsl:with-param name="result" select="$result"/>
            </xsl:call-template>
         </xsl:variable>
         <xsl:call-template name="ckbk:product-batcher">
               <xsl:with-param name="nodes"
                     select="$nodes[position() >= $batch-size]"/>
               <xsl:with-param name="result" select="$batch-product"/>
               <xsl:with-param name="batch-size" select="$batch-size"/>
         </xsl:call-template>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:product-dvc">
   <xsl:param name="nodes" select="/.."/>
   <xsl:param name="result" select="1"/>
   <xsl:param name="dvc-threshold" select="100"/>
   <xsl:choose>
      <xsl:when test="count($nodes) &lt;= $dvc-threshold">
         <xsl:call-template name="ckbk:product">
            <xsl:with-param name="nodes" select="$nodes"/>
            <xsl:with-param name="result" select="$result"/>
         </xsl:call-template>
      </xsl:when>
```

```
    <xsl:otherwise>
        <xsl:variable name="half" select="floor(count($nodes) div 2)"/>
        <xsl:variable name="product1">
           <xsl:call-template name="ckbk:product-dvc">
           <xsl:with-param name="nodes"
                select="$nodes[position() &lt;= $half]"/>
           <xsl:with-param name="result" select="$result"/>
           <xsl:with-param name="dvc-threshold" select="$dvc-threshold"/>
        </xsl:call-template>
     </xsl:variable>
     <xsl:call-template name="ckbk:product-dvc">
        <xsl:with-param name="nodes" select="$nodes[position() > $half]"/>
        <xsl:with-param name="result" select="$product1"/>
        <xsl:with-param name="dvc-threshold" select="$dvc-threshold"/>
     </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Лучший способ вычисления простых сумм – это встроенная в XPath функция sum(). Но, если надо вычислить сумму значений произвольной функции от узлов, то приходится делать одно из двух:

- применять рецепты из этого раздела либо
- сначала вычислить значения функции от всех узлов и сохранить результат в переменной в виде фрагмента дерева. Затем вызвать функцию-расширение для преобразования фрагмента в набор узлов, который можно подать на вход функции `sum`. В XSLT 2.0 обобщенное суммирование становится тривиальной задачей из-за исчезновения результирующих фрагментов дерева.

XSLT 2.0

Суммирование значений произвольной функции в версии 2.0 тривиально выполняется с помощью последовательностей, выражения `for` и функции XPath `sum`.

```
<!-- Сумма квадратов -->
<xsl:value select="sum(for $i in $nodes return $i * $i)"/>
```

Однако в XPath нет встроенной функции `prod()`, так что для перемножения чисел придется написать собственную:

```
<xsl:function name="ckbk:prod" as="xs:double">
  <xsl:param name="numbers" as="xs:double*"/>
  <xsl:sequence select="if (count($numbers) eq 0) then 0
                 else if (count($numbers) = 1) then $numbers[1]
                else $numbers[1] * ckbk:prod(subsequence($numbers,2))"/>
</xsl:function>
```

Обсуждение

XSLT 1.0

Методы разбиения на части и «разделяй и властвуй» оказываются полезны, когда нужно рекурсивно обработать потенциально большой набор узлов. Эксперименты показывают, что даже при наличии процессора XSLT, оптимизирующего хвостовую рекурсию, применение этих методов позволяет повысить производительность.

В главе 14 показано, как написать повторно используемую оснастку для запуска алгоритмов разбиения на части и «разделяй и властвуй».

XPath 2.0

При наличии функции `prod` возникает искушение вычислить факториал следующим образом:

```
<xsl:function name="ckbk:factorial" as="xs:double">
    <xsl:param name="n" as="xs:integer"/>
    <xsl:sequence select="if ($n eq 0) then 1 else ckbk:prod(1 to $n)"/>
</xsl:function>
```

Если `$n` мало, то это решение, может, и сгодится, но при увеличении `$n` накладные расходы на построение последовательности могут ухудшить производительность по сравнению с непосредственным решением. Об этом всегда следует помнить, прибегая к последовательностям. Майкл Кэй приводит и другие примеры в ходе обсуждения различий между решениями задачи о рыцарском турнире для версий 1.0 и 2.0 (*XSLT 2.0*, Third Edition, Wrox, 2004, стр. 753).

```
// что делает этот код?
<xsl:function name="ckbk:factorial" as="xs:decimal">
  <xsl:param name="n" as="xs:integer"/>
  <xsl:sequence select="if ($n eq 0) then 1 else $n * ckbk:factorial($n - 1)"/>
</xsl:function>
```

См. также

Димитр Новачев (Dimitre Novatchev) и Славомир Тышко (Slawomir Tyszko) сравнивают методы разбиения на части и «разделяй и властвуй» в статье на странице *http://www.vbxml.com/xsl/article/recurse/*.

3.7. Нахождение минимума и максимума

Задача

В наборе узлов требуется найти узел (или узлы) с минимальным (или максимальным) числовым значением.

Решение

XPath 1.0

В проекте EXSLT определены следующие функции для выполнения этих операций: `exsl:min`, `exsl:max`, `exsl:lowest` и `exsl:highest`. Функции `min` и `max` ищут узел с минимальным и максимальным числовым значением соответственно. В EXSLT функция `exsl:min` специфицирована следующим образом:

- Минимальное значение определяется так: набор узлов, переданный в качестве аргумента, сортируется в порядке возрастания, как это сделала бы команда `xsl:sort` в применении к типу данных `number`. Минимумом считается результат преобразования строкового значения первого узла в отсортированном списке в число с помощью функции `number`.
- Если набор узлов пуст или преобразование строкового значения любого узла в число дает результат `NaN`, возвращается `NaN`.

Функция `exsl:max` определяется аналогично. На сайте EXSLT предлагаются реализации на чистом XSLT, буквально отражающие эти определения (пример 3.8).

Пример 3.8. Реализации функций min и max, следующие непосредственно из определений EXSLT

```
<xsl:template name="ckbk:min">
  <xsl:param name="nodes" select="/.." />
  <xsl:choose>
    <xsl:when test="not($nodes)">NaN</xsl:when>
    <xsl:otherwise>
       <xsl:for-each select="$nodes">
           <xsl:sort data-type="number" />
           <xsl:if test="position() = 1">
             <xsl:value-of select="number(.)" />
           </xsl:if>
       </xsl:for-each>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:max">
  <xsl:param name="nodes" select="/.." />
  <xsl:choose>
     <xsl:when test="not($nodes)">NaN</xsl:when>
    <xsl:otherwise>
       <xsl:for-each select="$nodes">
          <xsl:sort data-type="number" order="descending" />
          <xsl:if test="position() = 1">
```

```
            <xsl:value-of select="number(.)" />
          </xsl:if>
        </xsl:for-each>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Возможно, вам непонятен смысл значения по умолчанию параметра `nodes (select="/..")`. Это просто идиоматический способ инициализировать пустой набор узлов (у корневого узла по определению не существует родителя).

Хотя в определениях `ckbk:min` и `ckbk:max` используется команда `xsl:sort`, реализацию не обязательно строить именно так. Если ваш процессор XSLT оптимизирует хвостовую рекурсию, то следующий вариант окажется более эффективным (пример 3.9).

Пример 3.9. Реализация функций min и max с помощью метода «разделяй и властвуй»

```
<xsl:template name="ckbk:max">
     <xsl:param name="nodes" select="/.."/>
     <xsl:param name="max"/>
  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:variable name="aNode" select="$nodes[ceiling($count div 2)]"/>
  <xsl:choose>
    <xsl:when test="not($count)">
      <xsl:value-of select="number($max))"/>
    </xsl:when>
    <xsl:when test="number($aNode) != number($aNode)">
      <xsl:value-of select="number($aNode))"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="ckbk:max">
      <xsl:with-param name="nodes"
           select="$nodes[not(. &lt;= number($aNode))]"/>
      <xsl:with-param name="max">
        <xsl:choose>
          <xsl:when test="not($max) or $aNode > $max">
            <xsl:value-of select="$aNode"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$max"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:with-param>
  </xsl:call-template>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:min">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="min"/>
  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:variable name="aNode" select="$nodes[ceiling($count div 2)]"/>
  <xsl:choose>
    <xsl:when test="not($count)">
      <xsl:value-of select="number($min))"/>
    </xsl:when>
    <xsl:when test="number($aNode) != number($aNode)">
      <xsl:value-of select="number($aNode))"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="ckbk:min">
        <xsl:with-param name="nodes"
           select="$nodes[not(. >= number($aNode))]"/>
        <xsl:with-param name="min">
          <xsl:choose>
             <xsl:when test="not($min) or $aNode &lt; $min">
               <xsl:value-of select="$aNode"/>
            </xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="$min"/>
             </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Как правило, приведенные выше реализации работают быстрее, чем основанные на команде `xsl:sort`. Но в некоторых вырожденных случаях они оказываются медленнее. Причина в том, что эффективность достигается за счет исключения из рассмотрения в среднем половины узлов на каждом шаге рекурсии. Однако возможна ситуация, когда на каждом проходе `aNode` является минимальным (в случае `ckbk:max`) или максимальным (в случае `ckbk:min`) узлом. Если это так, то при каждом рекурсивном вызове будет удаляться только один узел, и производительность окажется очень низкой. К счастью, данные чаще всего бывают либо предварительно упорядочены, либо случайны, а в этих случаях быстродействие будет на высоте.

Реализация EXSLT, основанная на `xsl:sort`, корректно обрабатывает нечисловые значения. Дело в том, что стандарт XSLT требует, чтобы в случае, когда тип данных равен `number`, нечисловые значения предшествовали числовым после сортировки.

Не пытайтесь использовать выражение `not(number($var))` для проверки на `NaN`! Я часто ловил себя на этой ошибке, которая на первый взгляд выглядит совершенно корректно. Функция `number` не проверяет, что аргумент является числом, а пытается преобразовать его в число. А это совсем не то, что вам нужно, – такая проверка сообщит, что 0 – не число, поскольку 0 преобразуется в `false`. Правильное условие выглядит так: `number($var) != number($var)`. Она работает, потому что `NaN` никогда не равно `NaN`, тогда как всякое число равно само себе. Не поддавайтесь искушению сократить это условие до `number($var) != $var`. Это будет работать только, если `$var` не является пустым набором узлов. Если хотите, можете применить совсем прямолинейный подход: `string(numer($var)) = 'NaN'`.

В EXSLT функция `ckbk:lowest` определена следующим образом:

- Функция `ckbk:lowest` возвращает те узлы из набора узлов, значения которых минимальны в данном наборе. Минимум по набору узлов вычисляется так же, как в функции `ckbk:min`. Узел имеет такое минимальное значение, если результат преобразования его строкового значения в число, как это делает функция `number`, равен минимальному значению, причем равенство определяется в терминах числового сравнения оператором =.
- Если хотя бы один узел в наборе имеет нечисловое значение, то функция `ckbk:min` возвращает NaN. Из определения сравнения чисел следует, что `NaN != NaN`. Поэтому, если хотя бы один узел в наборе имеет нечисловое значение, то функция `ckbk:lowest` возвращает пустой набор узлов.

Предлагаемая на сайте EXSLT реализация дословно следует этому определению, что может оказаться не слишком эффективно:

```
<xsl:template name="ckbk:lowest">
  <xsl:param name="nodes" select="/.."/>
  <xsl:id test="$nodes and not($nodes[number(.) != number(.)])">
    <xsl:variable name="min">
       <xsl:for-each select="$nodes">
          <xsl:sort data-type="number" />
          <xsl:if test="position() = 1">
           <xsl:value-of select="number(.)" />
          </xsl:if>
        </xsl:for-each>
     </xsl:variable>
     <xsl:copy-of select="$nodes[. = $min]" />
   </xsl:if>
</xsl:template>
```

Команда `xsl:if` проверяет, имеются ли в наборе узлы с нечисловыми значениями. Далее узлы сортируются для нахождения минимума, а затем собираются все узлы, значения которых совпадают с найденным минимумом. В примере 3.10 приведен код, в котором повторно используется функция `ckbk:min`, что делает сортировку ненужной.

Пример 3.10. Реализация `lowest` с повторным использованием `ckbk:min`

```
<xsl:template name="ckbk:lowest">
    <xsl:param name="nodes" select="/.."/>

    <xsl:variable name="min">
        <xsl:call-template name="ckbk:min">
           <xsl:with-param name="nodes" select="$nodes"/>
        </xsl:call-template>
    </xsl:variable>
    <xsl:choose>
        <xsl:when test="number($min) = number($min)">
           <xsl:copy-of select="$nodes[. = $min]"/>
        </xsl:when>
    </xsl:choose>
</xsl:template>
```

Наконец, `ckbk:lowest` можно реализовать с одним проходом по набору узлов, если вы готовы отказаться от повторного использования `ckbk:min` (пример 3.11).

Пример 3.11. Реализация `lowest` без использования `ckbk:min`

```
<xsl:template name="ckbk:lowest">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="lowest" select="/.."/>
  <xsl:variable name="index" select="ceiling(count($nodes) div 2)"/>
  <xsl:variable name="aNode" select="$nodes[$index]"/>
  <xsl:choose>
    <xsl:when test="not($index)">
      <xsl:copy-of select="$lowest"/>
    </xsl:when>
    <xsl:when test="number($aNode) != number($aNode)"/>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="not($lowest) or $aNode &lt; $lowest">
           <xsl:call-template name="ckbk:lowest">
            <xsl:with-param name="nodes" select="$nodes[not(. >= $aNode)]"/>
            <xsl:with-param name="lowest" select="$nodes[. = $aNode]"/>
        </xsl:call-template>
      </xsl:when>
```

```
        <xsl:when test="$aNode = $lowest">
            <xsl:call-template name="ckbk:lowest">
            <xsl:with-param name="nodes" select="$nodes[not(. >= $aNode)]"/>
            <xsl:with-param name="lowest" select="$lowest|$nodes[$index]"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
            <xsl:call-template name="ckbk:lowest">
              <xsl:with-param name="nodes" select="$nodes[not(. >= $aNode)]"/>
              <xsl:with-param name="lowest" select="$lowest"/>
            </xsl:call-template>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Интересно, что эта реализация работает хуже, возможно, из-за дополнительного копирования. В тестах, содержащих 10000 узлов с различным распределением данных (отсортированные в порядке возрастания, в порядке убывания, полуслучайные и случайные), реализация, основанная на `ckbk:min`, работала быстрее альтернативной в среднем на 40% (а часто и еще лучше). Рекурсивная реализация без использования `ckbk:min` оказалась на 24% медленнее.

Определение и реализации функции `ckbk:highest` получаются из `ckbk:lowest` обращением условий сравнения. Особо я их обсуждать не буду.

XSLT 2.0

В XPath 2.0 уже встроены функции `min` и `max`.

Обсуждение

Минимальное и максимальное значение в наборе узлов можно найти с помощью простых выражений XPath `<xsl:value-of select="$nodes[not($nodes < .)]"/>` и `<xsl:value-of select="$nodes[not($nodes > .)]"/>` соответственно. На обычном языке это означает: «Отобрать все узлы, для которых не существует узла с меньшим значением» и «Отобрать все узлы, для которых не существует узла с большим значением».

Однако вычислительная сложность этих, по видимости очень простых, выражений составляет $O(N^2)$, где N – количество узлов. Поэтому, если вы не уверены, что узлов будет заведомо немного, не прибегайте к такой идиоме. Впрочем, иногда другого выхода просто не остается, например, если нужно найти `min/max` внутри атрибута `select` команды `xsl:sort` или атрибута `use` команды `xsl:key` (для которых нельзя вызвать шаблон).

В одной из публикаций по XSLT предлагается следующая реализация отыскания минимумов, которая, как утверждается, более эффективна, чем основанная на `xsl:sort`.

```
<xsl:template name="ckbk:min">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="min" select="number('NaN')"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
      <xsl:value-of select="number($min)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="aNode" select="$nodes[1]"/>
      <xsl:call-template name="ckbk:min">
        <xsl:with-param name="nodes" select="$nodes[position() > 1]"/>
        <xsl:with-param name="min">
          <xsl:choose>
            <xsl:when test="$aNode &lt; $min or string($min) = 'NaN'">
              <xsl:value-of select="$aNode"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$min"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Но это не так ни в одной из реализаций XSLT, которые я тестировал, и я просто не могу себе представить, при каких условиях это было бы правдой. Причина, по которой эта реализация, скорее всего, будет работать медленнее, заключается в том, что на каждом шаге из рассмотрения исключается всего один узел. Даже при оптимизации хвостовой рекурсии число операций копирования узлов слишком велико. Можно легко впасть в заблуждение, думая, что такое рекурсивное решение так же эффективно, как итерация по индексированному массиву в языке C или Java. Однако увеличение индекса – совсем не то же самое, что создание нового набора узлов путем удаления из исходного начального элемента. А именно это мы и делаем в выражении `$nodes[position() > 1]`.

Во многих случаях, когда необходимо найти минимум, требуется также и максимум. Хорошо бы иметь функцию, которая дает оба значения по цене одного. Ее можно написать ценой небольшого усложнения:

```
<xsl:template name="ckbk:min-max">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="nodes-for-max" select="$nodes"/>
  <xsl:param name="min"/>
  <xsl:param name="max"/>
```

```
    <xsl:variable name="count1" select="count($nodes)"/>
    <xsl:variable name="aNode1" select="$nodes[ceiling($count1 div 2)]"/>
    <xsl:variable name="count2" select="count($nodes-for-max)"/>
    <xsl:variable name="aNode2" select="$nodes-for-max[ceiling($count2 div 2)]"/>
    <xsl:choose>
      <xsl:when test="not($count1) and not($count2)">
         <xsl:value-of select="concat(number($min),',',number($max))"/>
      </xsl:when>
      <xsl:when test="number($aNode1) != number($aNode1) and
                number($aNode2) != number($aNode2)">
        <xsl:value-of select="concat(number($aNode1),',',number($aNode2))"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:call-template name="ckbk:min-max">
          <xsl:with-param name="nodes"
             select="$nodes[not(. >= number($aNode1))]"/>
          <xsl:with-param name="nodes-for-max"
             select="$nodes-for-max[not(. &lt;= number($aNode2))]"/>
          <xsl:with-param name="min">
            <xsl:choose>
              <xsl:when test="not($min) or $aNode1 &lt; $min">
                  <xsl:value-of select="$aNode1"/>
               </xsl:when>
               <xsl:otherwise>
                  <xsl:value-of select="$min"/>
               </xsl:otherwise>
            </xsl:choose>
          </xsl:with-param>
          <xsl:with-param name="max">
            <xsl:choose>
              <xsl:when test="not($max) or $aNode2 > $max">
                <xsl:value-of select="$aNode2"/>
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="$max"/>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Тестирование показывает, что и этот подход превосходит по скорости решение, основанное на сортировке.

При обсуждении минимумов и максимумов мы рассматривали только один частный случай: когда узлы содержат числа. Более общая задача – найти минимум и максимум из значений функции, определенной на узлах. Возьмем, к примеру, случай, когда есть множество положительных и отрицательных чисел, а нужно найти минимальный квадрат значения. Хотя эту задачу легко решить путем небольшой модификации показанного выше кода, хотелось бы иметь единую повторно используемую реализацию. В главе 14 мы вернемся к этой проблеме и опишем несколько способов создания обобщенных решений.

3.8. Вычисление статистических функций

Задача

Требуется вычислить среднее, дисперсию и стандартное отклонение.

Решение

XSLT 1.0

В математической статистике находят применение три вида средних значений: среднее арифметическое, медиана и мода.

Среднее арифметическое вычисляется тривиально – находим сумму, пользуясь рецептом 3.6, и делим ее на количество слагаемых.

Медиана – это число, которое оказывается в середине набора после его сортировки. Если количество членов набора четно, то берется среднее двух чисел, оказавшихся в середине.

```
<xsl:template name="ckbk:median">
  <xsl:param name="nodes" select="/.."/>
  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:variable name="middle" select="ceiling($count div 2)"/>
  <xsl:variable name="even" select="not($count mod 2)"/>

  <xsl:variable name="m1">
    <xsl:for-each select="$nodes">
       <xsl:sort data-type="number"/>
       <xsl:if test="position() = $middle">
          <xsl:value-of select=". + ($even * ./following-sibling::*[1])"/>
       </xsl:if>
    </xsl:for-each>
  </xsl:variable>

  <!-- Медиана -->
  <xsl:value-of select="$m1 div ($even + 1)"/>
</xsl:template>
```

Случай четного количества узлов обрабатывается с помощью уже встречавшегося нам трюка с преобразованием булевской величины в число. Если количество

узлов нечетно, то $m1 в конце концов оказывается средним узлом, и для получения ответа мы делим его на 1. Если же количество узлов четно, то на последнем шаге $m1 будет равно сумме двух средних узлов, и эта величина делится на 2.

Мода – это элемент (или элементы), наиболее часто встречающийся в множестве, которое не обязательно состоит из чисел. Если одинаковость узлов можно установить сравнением их строковых значений, то годится следующее решение:

```
<xsl:template name="ckbk:mode">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="max" select="0"/>
  <xsl:param name="mode" select="/.."/>

  <xsl:choose>
    <xsl:when test="not($nodes)">
      <xsl:copy-of select="$mode"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="first" select="$nodes[1]"/>
    <xsl:variable name="try" select="$nodes[. = $first]"/>
      <xsl:variable name="count" select="count($try)"/>
      <!-- Рекурсия по узлам, не равным первому -->
      <xsl:call-template name="ckbk:mode">
        <xsl:with-param name="nodes" select="$nodes[not(. = $first)]"/>
        <!-- Если мы нашли узел, встречающийся чаще
         других, передаем count, иначе прежнее значение
         счетчика максимального числа вхождений -->
        <xsl:with-param name="max"
           select="($count > $max) * $count + not($count > $max) * $max"/>
        <!-- Вычисляем новую моду ... -->
        <xsl:with-param name="mode">
          <xsl:choose>
            <!-- первый элемент в try, если найден новый максимум -->
            <xsl:when test="$count > $max">
              <xsl:copy-of select="$try[1]"/>
            </xsl:when>
            <!-- старая мода объединяется с первым элементом в try, если
           число вхождений равно текущему максимуму -->
            <xsl:when test="$count = $max">
           <!-- Caution: you will need to convert $mode to a -->
           <!-- node set if you are using a version of XSLT -->
           <!-- that does not convert automatically -->
              <xsl:copy-of select="$mode | $try[1]"/>
            </xsl:when>
            <!-- в противном случае старая мода не изменяется -->
```

```
            <xsl:otherwise>
              <xsl:copy-of select="$mode"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:with-param>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Если это не так, заменим сравнение подходящей проверкой. Например, если эквивалентность означает равенство атрибута `age`, то проверка выглядит так: `./@age = $first/age`.

Дисперсия и стандартное отклонение применяются в статистике для оценки разброса значений относительно среднего. Простейший способ вычисления дисперсии основан на вычислении трех величин: `sum` – сумма всех чисел, `sum-sq` – сумма квадратов чисел и `count` – количество чисел. Тогда дисперсия равна `(sum-sq - sum2 / count) / count -1`. Все это можно вычислить в одном шаблоне с хвостовой рекурсией:

```
<xsl:template name="ckbk:variance">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="sum" select="0"/>
  <xsl:param name="sum-sq" select="0"/>
  <xsl:param name="count" select="0"/>
  <xsl:choose>
    <xsl:when test="not($nodes)">
      <xsl:value-of select="($sum-sq - ($sum * $sum) div $count)
                            div ($count - 1)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:variable name="value" select="$nodes[1]"/>
      <xsl:call-template name="ckbk:variance">
        <xsl:with-param name="nodes" select="$nodes[position() != 1]"/>
        <xsl:with-param name="sum" select="$sum + $value"/>
        <xsl:with-param name="sum-sq" select="$sum-sq + ($value * $value)"/>
        <xsl:with-param name="count" select="$count + 1"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Вероятно, вы заметили, что этот шаблон – вариация на тему `ckbk:sum`, мы просто включили в него вычисление еще двух компонентов дисперсии. Поэтому, если процессор XSLT не поддерживает хвостовую рекурсию, то на больших наборах данных неизбежны проблемы. В таком случае придется обратиться к другой стратегии,

основанной на общепринятом определении дисперсии: $\sum(mean - x_i)^2 / (count - 1)$. Сначала методом «разделяй и властвуй» или разбиения на части мы вычисляем сумму элементов, и делением на `count` получаем среднее. Затем таким же образом вычисляется сумма квадратов разностей между каждым элементом и средним. И, наконец, делим результат на `count - 1`.

Зная дисперсию, вычислить стандартное отклонение тривиально – нужно лишь извлечь квадратный корень. Об извлечении корня см. рецепт 3.5.

XSLT 2.0

```
<!-- Медиана -->
<xsl:function name="ckbk:median">
  <xsl:param name="nodes" as="xs:double*" />

  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:variable name="middle" select="ceiling($count div 2)"/>

  <xsl:variable name="sorted" as="xs:double*">
    <xsl:perform-sort select="$nodes">
       <xsl:sort data-type="number"/>
    </xsl:perform-sort>
  </xsl:variable>

<!-- Мода -->
<xsl:function name="ckbk:mode" as="item()*">
  <xsl:param name="nodes" as="item()*"/>
  <!-- Сначала находим уникальные значения -->
  <xsl:variable name="distinct" select="distinct-values($nodes)" as="item()*"/>
    <!-- Получаем последовательность счетчиков числа вхождений каждого
    уникального значения -->
    <xsl:variable name="counts"
                  select="for $i in $distinct return count($nodes[. = $i])"
                  as="xs:integer*"/>
  <!-- Находим максимальный счетчик -->
  <xsl:variable name="max" select="max($counts)" as="xs:integer?"/>
  <!-- Возвращаем значения, встречающиеся максимальное число раз -->
  <xsl:sequence select="$distinct[position() = index-of($counts,$max)]"/>
</xsl:function>

<!-- Дисперсия -->
<xsl:function name="ckbk:variance" as="xs:double">
  <xsl:param name="nodes" as="xs:double*"/>
  <xsl:variable name="sum" select="sum($nodes)"/>
  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:sequence select="if ($count lt 2)
```

```
                then 0
                 else sum(for $i in $nodes return $i * $i) -
                      ($sum * $sum) div
                          ($count * $count - $count)"/>
  </xsl:function>
```

Обсуждение

XSLT 1.0

Статистические функции повсеместно применяются для анализа числовых данных, так что приведенные выше шаблоны окажутся полезным добавлением в копилку ваших инструментов. Однако XSLT никогда не задумывался как язык для статистического анализа. Альтернативный подход состоит в том, чтобы воспользоваться XSLT для предварительного преобразования данных из формата XML в формат, где значения разделены запятыми или символами табуляции, с последующим импортом файла в электронную таблицу или специализированный пакет статистических программ.

XSLT 2.0

Как и раньше, в версии XSLT 2.0 решение записывается значительно проще. Еще важнее тот факт, что реализация моды в этой версии гораздо быстрее и компактнее, да и отлаживать ее легче. Помнится, у меня ушло не меньше двух часов на то, чтобы написать правильный код для первого издания этой книги. А в версии 2.0 потребовалось всего-то 15 минут после того, как я понял, что можно воспользоваться функцией `distinct-values`.

3.9. Вычисление комбинаторных функций

Задача

Требуется вычислить число перестановок или сочетаний размера r для данного множества.

Решение

XSLT 1.0

Если вы знаете, что число перестановок размера r равно `N! / r!`, а число сочетаний – `N! / r! * (N-r)!`, то может возникнуть искушение пропустить этот пример, поскольку ранее уже был приведен способ вычисления факториала. Однако так как факториалы растут очень быстро, придется проявить некоторое хитроумие.

```
<xsl:template name="ckbk:P">
  <xsl:param name="n" select="1"/>
  <xsl:param name="r" select="1"/>
    <xsl:choose>
       <xsl:when test="$n &lt; 0 or $r &lt; 0">NaN</xsl:when>
```

```
        <xsl:when test="$n = 0">0</xsl:when>
        <xsl:otherwise>
            <xsl:call-template name="prod-range">
            <xsl:with-param name="start" select="$r + 1"/>
            <xsl:with-param name="end" select="$n"/>
            </xsl:call-template>
         </xsl:otherwise>
       </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:C">
    <xsl:param name="n" select="1"/>
    <xsl:param name="r" select="1"/>
      <xsl:choose>
         <xsl:when test="$n &lt; 0 or $r &lt; 0">NaN</xsl:when>
         <xsl:when test="$n = 0">0</xsl:when>
         <xsl:otherwise>
            <xsl:variable name="min"
             select="($r &lt;= $n - $r) * $r +  ($r > $n - $r) * $n - $r"/>
          <xsl:variable name="max"
             select="($r >= $n - $r) * $r + ($r &lt; $n - $r) * $n - $r"/>
          <xsl:variable name="numerator">
            <xsl:call-template name="prod-range">
              <xsl:with-param name="start" select="$max + 1"/>
              <xsl:with-param name="end" select="$n"/>
            </xsl:call-template>
          </xsl:variable>
          <xsl:variable name="denominator">
            <xsl:call-template name="ckbk:fact">
               <xsl:with-param name="number" select="$min"/>
            </xsl:call-template>
          </xsl:variable>
          <xsl:value-of select="$numerator div $denominator"/>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

XSLT 2.0

```
<xsl:function name="ckbk:P" as="xs:decimal">
    <xsl:param name="r" as="xs:integer"/>
    <xsl:param name="n" as="xs:integer"/>
    <xsl:sequence select="if ($n eq 0)
                          then 0 else ckbk:prod-range($r + 1, $n)"/>
</xsl:function>
```

```
    <xsl:function name="ckbk:C" as="xs:decimal">
     <xsl:param name="r" as="xs:integer"/>
     <xsl:param name="n" as="xs:integer"/>
     <xsl:variable name="min" select="min( ($r, $n - $r) )" as="xs:integer"/>
     <xsl:variable name="max" select="max( ($r, $n - $r) )" as="xs:integer"/>
     <xsl:sequence select="if ($n eq 0) then 0
                           else ckbk:prod-range($max + 1, $n) div
                                ckbk:factorial($min)"/>
</xsl:function>
```

Обсуждение

Основная идея этих решений – уменьшить число операций умножения. Когда факториал большего числа делится на факториал меньшего, то многие сомножители сокращаются. Поэтому для реализации лучше воспользоваться функцией `prod-range` (рецепт 3.5), а не `factorial`. Вычисление числа сочетаний чуть сложнее, так как нужно выбрать максимум из r и (n-r), чтобы сократить возможно большее число сомножителей.

3.10. Проверка битов

Задача

Требуется рассматривать числа как битовые маски, хотя в языке XSLT нет понятия целого числа a, значит, нет и поразрядных операций.

При работе с XML не пытайтесь во чтобы бы то ни стало закодировать информацию в виде набора битов. Применяйте это решение лишь тогда, когда не можете контролировать способ представления данных.

Решение

Следующее решение работает для 16-разрядных чисел, но легко обобщается на 32-разрядные:

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

<!-- степени двойки -->
<xsl:variable name="bit15" select="32768"/>
<xsl:variable name="bit14" select="16384"/>
<xsl:variable name="bit13" select="8192"/>
<xsl:variable name="bit12" select="4096"/>
<xsl:variable name="bit11" select="2048"/>
```

```
<xsl:variable name="bit10" select="1024"/>
<xsl:variable name="bit9"  select="512"/>
<xsl:variable name="bit8"  select="256"/>
<xsl:variable name="bit7"  select="128"/>
<xsl:variable name="bit6"  select="64"/>
<xsl:variable name="bit5"  select="32"/>
<xsl:variable name="bit4"  select="16"/>
<xsl:variable name="bit3"  select="8"/>
<xsl:variable name="bit2"  select="4"/>
<xsl:variable name="bit1"  select="2"/>
<xsl:variable name="bit0"  select="1"/>

<xsl:template name="bitTest">
  <xsl:param name="num"/>
  <xsl:param name="bit" select="$bit0"/>
  <xsl:choose>
    <xsl:when test="( $num mod ( $bit * 2 ) ) -
                    ( $num mod ( $bit     ) )">1</xsl:when>
    <xsl:otherwise>0</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bitAnd">
  <xsl:param name="num1"/>
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="test" select="$bit15"/>

  <xsl:variable name="nextN1"
    select="($num1 >= $test) * ($num1 - $test) + not($num1 >= $test) * $num1"/>
  <xsl:variable name="nextN2"
    select="($num2 >= $test) * ($num2 - $test) + not($num2 >= $test) * $num2"/>

  <xsl:choose>
    <xsl:when test="$test &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num1 >= $test and $num2 >= $test">
      <xsl:call-template name="bitAnd">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result + $test"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
```

```
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitAnd">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bitOr">
  <xsl:param name="num1"/>
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="test" select="$bit15"/>

  <xsl:variable name="nextN1"
    select="($num1 >= $test) * ($num1 - $test) + not($num1 >= $test) * $num1"/>
  <xsl:variable name="nextN2"
    select="($num2 >= $test) * ($num2 - $test) + not($num2 >= $test) * $num2"/>

  <xsl:choose>
    <xsl:when test="$test &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num1 >= $test or $num2 >= $test">
      <xsl:call-template name="bitOr">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result + $test"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitOr">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
```

```
</xsl:template>

<xsl:template name="bitXor">
  <xsl:param name="num1"/>
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="test" select="$bit15"/>

  <xsl:variable name="nextN1"
    select="($num1 >= $test) * ($num1 - $test) + not($num1 >= $test) * $num1"/>
  <xsl:variable name="nextN2"
    select="($num2 >= $test) * ($num2 - $test) + not($num2 >= $test) * $num2"/>

  <xsl:choose>
    <xsl:when test="$test &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num1 >= $test and not($num2 >= $test)
        or not($num1 >= $test) and $num2 >= $test">
      <xsl:call-template name="bitXor">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result + $test"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitXor">
        <xsl:with-param name="num1" select="$nextN1"/>
        <xsl:with-param name="num2" select="$nextN2"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bitNot">
  <xsl:param name="num"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="test" select="$bit15"/>

  <xsl:choose>
    <xsl:when test="$test &lt; 1">
```

```
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num >= $test">
      <xsl:call-template name="bitNot">
        <xsl:with-param name="num" select="$num - $test"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitNot">
        <xsl:with-param name="num" select="$num"/>
        <xsl:with-param name="result" select="$result + $test"/>
        <xsl:with-param name="test" select="$test div 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Этот способ проверки битов (шаблон bitTest), основанный на арифметике по модулю, обсуждался на конференции XML DevCon, состоявшейся в 2001 году в Лондоне. Мы реализовали поразрядные логические операции рекурсивно с помощью операций сравнения и вычитания, но можно было бы воспользоваться операциями деления и взятия остатка, как показано ниже:

```
<xsl:template name="bitAnd">
  <xsl:param name="num1"/>
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="pow2" select="$bit0"/>

  <xsl:choose>
    <xsl:when test="$num1 &lt; 1 or $num2 &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num1 mod 2 and $num2 mod 2">
      <xsl:call-template name="bitAnd">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result + $pow2"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
```

```
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitAnd">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bitOr">
  <xsl:param name="num1"/>
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="pow2" select="$bit0"/>

  <xsl:choose>
    <xsl:when test="$num1 &lt; 1 and $num2 &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="boolean($num1 mod 2) or boolean($num2 mod 2)">
      <xsl:call-template name="bitOr">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result + $pow2"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitOr">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="bitXor">
  <xsl:param name="num1"/>
```

```
  <xsl:param name="num2"/>
  <xsl:param name="result" select="0"/>
  <xsl:param name="pow2" select="$bit0"/>

  <xsl:choose>
    <xsl:when test="$num1 &lt; 1 and $num2 &lt; 1">
      <xsl:value-of select="$result"/>
    </xsl:when>
    <xsl:when test="$num1 mod 2 + $num2 mod 2 = 1">
      <xsl:call-template name="bitXor">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result + $pow2"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="bitXor">
        <xsl:with-param name="num1" select="floor($num1 div 2)"/>
        <xsl:with-param name="num2" select="floor($num2 div 2)"/>
        <xsl:with-param name="result" select="$result"/>
        <xsl:with-param name="pow2" select="$pow2 * 2"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

XSLT 2.0

Найти решение для версии 2.0 оставляю читателю в качестве упражнения. Как и в других рецептах, следует преобразовать шаблоны в функции и заменить идиоматические выражения для работы с булевскими величинами – `($num1 >= $test) * ($num1 - $test) + not($num1 >= $test) * $num1` – выражениями if-then-else.

Глава 4. Даты и время

Знает ли кто-нибудь, что такое время?
А есть ли кому-нибудь до этого дело?
Чикаго

4.0. Введение

В самом языке XSLT 1.0 нет понятия времени. Однако дата и время – немаловажный аспект повседневной жизни. Необходимость манипулировать ими возникает в программах очень часто, особенно при разработке для Web. Поэтому так странно и печально, что в XSLT 1.0 не встроена поддержка для работы с ними.

В версии 2.0 ситуация изменилась к лучшему. В XPath 2.0 включены многочисленные функции для манипулирования датами, временем и временными интервалами. В общем, единственное, чего можно было бы пожелать, – это чтобы функции имели не такие длинные имена! Мне говорили, что имена вроде `subtract-dates-yielding-yearMonthDuration()` были введены отчасти для того, чтобы умиротворить разработчиков языка XQuery. Совместное использование двух стилей записи имен – с черточкой-разделителем и заглавными буквами в начале слова – объясняется тем, что комитеты по разработке XPath и XML Schema приняли различные соглашения.

Примеры в этой главе показывают, как можно компенсировать отсутствие поддержки даты и времени в XSLT 1.0. К сожалению, невозможно реализовать одну из самых важных функций – получение текущей даты и времени. Для этого придется обратиться к какому-нибудь другому языку, в котором реализовано взаимодействие с аппаратным таймером. Такой механизм имеется и в Java, и в JavaScript. Если же вам нужно лишь отформатировать дату или время, которые уже имеются в документе, то приведенные ниже шаблоны позволят решить большинство возникающих задач. В версии 2.0 имеются функции `current-date()`, `current-time()` и `current-dateTime()`.

Манипуляции с датами и временем без поддержки со стороны языка – дело непростое, но в общем-то сводится к арифметическим операциям над целыми числами и, главным образом, к преобразованиям из одной системы счисления в другую. Для обращения с календарями, отличными от григорианского, нужно еще обладать знаниями из области истории, религии и культуры. Читатели, которым процедуры манипулирования датой и временем без надобности, могут спокойно пропустить эту главу, так как в ней почти нет новых приемов работы с XSLT. Тем же, кому интересны теоретические основы вычислений, стоит обратиться к источникам, упомянутым в разделе «См. также» ниже.

Я благодарен Джейсону Даймонду (Jason Diamond), который любезно предоставил многие шаблоны, относящиеся к григорианскому календарю. Код для других календарей – это перевод на XSLT открытой реализации, написанной Эдвардом М. Рейнгольдом (Edward M. Reingold) на языке Lisp. Некоторые алгоритмы были изменены, чтобы лучше соответствовать структуре XSLT и коду Джейсона, положенному в основание всей библиотеки.

Пусть вас не пугает способ передачи даты в большинство шаблонов. Он выбран для удобства. Дату можно передать двумя путями: в виде строки, форматированной согласно правилам ISO для представления даты и времени, либо в виде отдельных компонентов, соответствующих году, месяцу и дню. Оба варианта продемонстрированы ниже:

```
<xsl:call-template name="ckbk:calculate-day-of-the-week">
    <xsl:with-param name="date-time" select="'2002-01-01T01:00:00'"/>
</xsl:call-template>

<xsl:call-template name="ckbk:calculate-day-of-the-week">
    <xsl:with-param name="date" select="'2002-01-01'"/>
</xsl:call-template>

<xsl:call-template name="ckbk:calculate-day-of-the-week">
    <xsl:with-param name="year" select="2002"/>
    <xsl:with-param name="month" select="01"/>
    <xsl:with-param name="day" select="01"/>
</xsl:call-template>
```

Во всех случаях вычисляется день недели, на который приходится дата 1 января 2002 года. Первые два способа удобны, если дата уже представлена в формате ISO, а последний – если дата хранится в виде отдельных компонентов. Можно также переопределить части ISO-представления:

```
<xsl:call-template name="ckbk:calculate-day-of-the-week">
    <xsl:with-param name="date" select="'2002-01-01'"/>
    <xsl:with-param name="day" select="25"/>
</xsl:call-template>
```

Если явно не оговорено противное, во всех шаблонах предполагается григорианский календарь, который используется в большинстве стран Запада.

См. также

В FAQe по календарям Клауса Тендеринга (Claus Tøndering) (*http://www.pauahtun.org/CalendarFAQ/cal/calendar24.pdf*) приводятся теоретические основы многих вычислений, встречающихся в этой главе.

Оригинальный XSLT-код Джейсона Даймонда можно найти по адресу *http://xsltsl.sourceforge.net/date-time.html*.

На странице *http:/emr.cs.iit.edu/~reingold/calendar.ps* Нахума Дершовица (Nachum Dershowitz) и Эдварда М. Рейнгольда вы найдете дополнительные материалы на

темы вычислений в различных календарях. В этой статье обсуждаются упоминаемые ниже не-григорианские системы.

В этой главе не рассматриваются китайский и индийский календари, поскольку они гораздо сложнее. Информацию о лежащей в их основе математике можно найти на страницах *http://www.math.nus.edu.sg/aslaksen/calendar/chinese.shtml* и *http://www.math.nus.edu.sg/aslaksen/calendar/indian.shtml* соответственно.

Если по какой-то причине вам понадобится календарь майя, французской революции или старый индуистский, то обратитесь к упомянутому выше сайту Calendrical Calculations по адресу *http:/emr.cs.iit.edu/~reingold/calendar.ps*.

В проект EXSLT.org включен модуль для работы с датой и временем, предлагающий практически ту же функциональность, что и «григорианские шаблоны», приведенные в этой главе. Дополнительно предлагаются шаблоны для работы с временными интервалами.

В спецификации XForms (*http://www.w3.org/MarkUp/Forms/*) описываются некие средства, относящиеся к дате и времени. Используются лексические представления в формате ISO на основе XML Schema (интервалы подразделяются на типы `dateTime` и `yearMonth`). Введены такие новые функции: `now()`, `days-from-date()`, `seconds-from-dateTime()`, `seconds()` и `months()`.

4.1. Вычисление дня недели

Задача

Зная день, месяц и год, требуется определить день недели.

Решение

XSLT 1.0

Следующий код возвращает целое число в диапазоне от 0 до 6, где 0 соответствует воскресенью.

```
<xsl:template name="ckbk:calculate-day-of-the-week">
    <xsl:param name="date-time"/>
    <xsl:param name="date" select="substring-before($date-time,'T')"/>
    <xsl:param name="year" select="substring-before($date,'-')"/>
    <xsl:param name="month"
            select="substring-before(substring-after($date,'-'),'-')"/>
    <xsl:param name="day"
            select="substring-after(substring-after($date,'-'),'-')"/>

    <xsl:variable name="a" select="floor((14 - $month) div 12)"/>
    <xsl:variable name="y" select="$year - $a"/>
    <xsl:variable name="m" select="$month + 12 * $a - 2"/>
    <xsl:value-of select="($day + $y + floor($y div 4) - floor($y div 100)
    + floor($y div 400) + floor((31 * $m) div 12)) mod 7"/>

</xsl:template>
```

XSLT 2.0

Функция `format-date` позволяет получить день недели в виде числа или зависящей от языка строки.

Обсуждение

И в этом, и в других рецептах применяется функция `floor()` из языка XPath. Это единственный способ эмулировать целочисленные вычисления в XSLT 1.0, где все числа представлены в формате с плавающей точкой. То, что подобные вычисления работают правильно, объясняется особенностями григорианского календаря, никак не связанными с XSLT. Например, из любых 400 последовательных годов 97 являются високосными, поэтому високосным является каждый год, который делится на 4, кроме тех, что делятся на 100, но не делятся на 400. Отсюда и последнее выражение в приведенном выше шаблоне. Дополнительную информацию см. во врезке «Логика, лежащая в основе математики».

ЛОГИКА, ЛЕЖАЩАЯ В ОСНОВЕ МАТЕМАТИКИ

Клаус Тендеринг любезно приподнял завесу тайны, скрывающейся за этими вычислениями. Сначала вычисляются следующие значения:

```
a = (14 - month) / 12
y = year - a
m = month + 12*a - 2
```

Смысл их в том, чтобы перенести начало года с 1 января на 1 марта. Поскольку a равно 1 для января и февраля и 0 для всех остальных месяцев, то для этих двух месяцев из года вычитается 1. Таким образом, m принимает значение 1 для марта, 2 для апреля,... 10 для декабря, 11 для января и 12 для февраля.

Теперь год начинается 1 марта, и это дает нам два преимущества. Во-первых, дополнительный день приходится на конец года, что упрощает дальнейшие вычисления. А во-вторых, это означает, что число дней в месяце определяется простой схемой:

```
31 30 31 30 31 (март-июль)
31 30 31 30 31 (август-декабрь)
31 X (январь-февраль)
```

Это дает нам возможность вычислить смещение дня от начала недели по формуле (31*m)/12.

4.2. Вычисление последнего дня месяца

Задача

Зная месяц и год, вычислить последний день месяца.

Решение

XSLT 1.0

```
<xsl:template name="ckbk:last-day-of-julian-month">
  <xsl:param name="month"/>
  <xsl:param name="year"/>
  <xsl:choose>
    <xsl:when test="$month = 2 and
        not($year mod 4) and
        ($year mod 100 or not($year mod 400))">
      <xsl:value-of select="29"/>
        </xsl:when>
    <xsl:otherwise>
       <xsl:value-of
         select="substring('312831303130313130313031',
                           2 * $month - 1,2)"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

XSLT 2.0

Можно преобразовать шаблон из рецепта для версии 1.0 в функцию или, воспользовавшись функциями для работы с датами, вычесть единицу из первого дня следующего месяца. Второе решение проще понять, но первое, вероятно, будет работать быстрее.

```
<xsl:function name="ckbk:last-day-of-month" as="xs:integer">

  <xsl:param name="month" as="xs:integer"/>
  <xsl:param name="year" as="xs:integer"/>

  <!-- Первый день месяца в данном году -->
  <xsl:variable name="date"
          select="xs:date(concat(xs:string($year),
                         '-',format-number($month,'00'),'-01'))"/>
         as="xs:date"/>

    <xsl:variable name="m"
           select="xdt:yearMonthDuration('P1M')"
           as="xdt:yearMonthDuration"/>

    <xsl:variable name="d"
           select="xdt:dayTimeDuration('P1D')"
           as="xdt:dayTimeDuration"/>
```

```
    <xsl:sequence select="day-from-date(($date + $m) - $d)"/>

</xsl:function>
```

Обсуждение

Этой функцией можно воспользоваться для построения страниц календаря. Зная правила вычисления високосных лет, легко понять, как она работает. Код получен трансляцией с Lisp, где число дней в каждом месяце извлекалось из списка. Я решил для той же цели применить функцию `substring`. Возможно, вам больше нравится несколько элементов `when`. Можно также хранить данные о каждом месяце, в частности, о количестве дней в нем, в XML-документе.

См. также

О способах хранения метаданных о календаре в XML-документе см. рецепт 4.5.

4.3. Получение названий дней и месяцев

Задача

Требуется преобразовать числовые значения дней недели и месяцев в символьные.

Решение

XSLT 1.0

Если для вашего приложения интернационализация не нужна, подойдет такой простой код:

```
<xsl:template name="ckbk:get-day-of-the-week-name">
  <xsl:param name="day-of-the-week"/>

  <xsl:choose>
    <xsl:when test="$day-of-the-week = 0">Воскресенье</xsl:when>
    <xsl:when test="$day-of-the-week = 1">Понедельник</xsl:when>
    <xsl:when test="$day-of-the-week = 2">Вторник</xsl:when>
    <xsl:when test="$day-of-the-week = 3">Среда</xsl:when>
    <xsl:when test="$day-of-the-week = 4">Четверг</xsl:when>
    <xsl:when test="$day-of-the-week = 5">Пятница</xsl:when>
    <xsl:when test="$day-of-the-week = 6">Субботу</xsl:when>
    <xsl:otherwise>
          ошибка: <xsl:value-of select="$day-of-the-week"/>
      </xsl:otherwise>
  </xsl:choose>
```

```
</xsl:template>

<xsl:template name="ckbk:get-day-of-the-week-abbreviation">
  <xsl:param name="day-of-the-week"/>

  <xsl:choose>
    <xsl:when test="$day-of-the-week = 0">Вс</xsl:when>
    <xsl:when test="$day-of-the-week = 1">Пн</xsl:when>
    <xsl:when test="$day-of-the-week = 2">Вт</xsl:when>
    <xsl:when test="$day-of-the-week = 3">Ср</xsl:when>
    <xsl:when test="$day-of-the-week = 4">Чт</xsl:when>
    <xsl:when test="$day-of-the-week = 5">Пт</xsl:when>
    <xsl:when test="$day-of-the-week = 6">Сб</xsl:when>
    <xsl:otherwise>
       ошибка: <xsl:value-of select="$day-of-the-week"/>
      </xsl:otherwise>
  </xsl:choose>

</xsl:template>

<xsl:template name="ckbk:get-month-name">
  <xsl:param name="month"/>

  <xsl:choose>
    <xsl:when test="$month = 1">Январь</xsl:when>
    <xsl:when test="$month = 2">Февраль</xsl:when>
    <xsl:when test="$month = 3">Март</xsl:when>
    <xsl:when test="$month = 4">Апрель</xsl:when>
    <xsl:when test="$month = 5">Май</xsl:when>
    <xsl:when test="$month = 6">Июнь</xsl:when>
    <xsl:when test="$month = 7">Июль</xsl:when>
    <xsl:when test="$month = 8">Август</xsl:when>
    <xsl:when test="$month = 9">Сентябрь</xsl:when>
    <xsl:when test="$month = 10">Октябрь</xsl:when>
    <xsl:when test="$month = 11">Ноябрь</xsl:when>
    <xsl:when test="$month = 12">Декабрь</xsl:when>
    <xsl:otherwise>ошибка: <xsl:value-of select="$month"/></xsl:otherwise>
  </xsl:choose>

</xsl:template>

<xsl:template name="ckbk:get-month-abbreviation">
  <xsl:param name="month"/>
```

```
  <xsl:choose>
    <xsl:when test="$month = 1">Янв</xsl:when>
    <xsl:when test="$month = 2">Фев</xsl:when>
    <xsl:when test="$month = 3">Мар</xsl:when>
    <xsl:when test="$month = 4">Апр</xsl:when>
    <xsl:when test="$month = 5">Май</xsl:when>
    <xsl:when test="$month = 6">Июн</xsl:when>
    <xsl:when test="$month = 7">Июл</xsl:when>
    <xsl:when test="$month = 8">Авг</xsl:when>
    <xsl:when test="$month = 9">Сен</xsl:when>
    <xsl:when test="$month = 10">Окт</xsl:when>
    <xsl:when test="$month = 11">Ноя</xsl:when>
    <xsl:when test="$month = 12">Дек</xsl:when>
    <xsl:otherwise>ошибка: <xsl:value-of select="$month"/></xsl:otherwise>
  </xsl:choose>

</xsl:template>
```

XSLT 2.0

В XSLT 2.0 есть функция `format-date()`, которая решает как рассматриваемую в этом рецепте, так и многие другие задачи. В некоторых реализациях она поддерживает и интернационализацию.

```
<xsl:function name="ckbk:get-day-of-the-week-name" as="xs:string">
  <xsl:param name="day-of-the-week" as="xs:integer"/>

  <!-- Подойдет любое прошедшее воскресенье. Я произвольно выбрал
  воскресенье 14 августа 2005, так как именно в этот день я пишу
  данный текст -->

  <xsl:variable name="date"
    select="xs:date('2005-08-'). string(14 + $day-of-the-week))" as="xs:date"/>

  <xsl:sequence select="format-date($date, "[F]"/>
</xsl:function>

<xsl:function name="ckbk:get-day-of-the-week-name-abbr" as="xs:string">
  <xsl:param name="day-of-week" as="xs:integer"/>

  <xsl:variable name="date"
    select="xs:date('2005-08-'). string(14 + $day-of-the-week))" as="xs:date"/>

  <xsl:sequence select="format-date($date, '[FNn,3-3]')"/>

</xsl:function>
```

```
<xsl:function name="ckbk:get-month-name" as="xs:string">
  <xsl:param name="month" as="xs:integer"/>

  <xsl:variable name="jan01" select="xs:date('2005-01-01')" as="xs:date"/>

  <!-- Здесь для получения даты, передаваемой format-date, мы выполняем
     операцию сложения даты с интервалом, а не манипулируем строками -->
  <xsl:sequence select="format-date($jan01 +
       xdt:yearMonthDuration(concat('P',$month - 1,'M')), '[MNn]')"/>

</xsl:function>

<xsl:function name="ckbk:get-month-name-abbr" as="xs:string">
  <xsl:param name="month" as="xs:integer"/>

  <xsl:variable name="jan01" select="xs:date('2005-01-01')" as="xs:date"/>

  <xsl:sequence select="format-date($jan01 +
      xdt:yearMonthDuration(concat('P',$month - 1,'M')), '[MNn,3-3]')"/>

</xsl:function>
```

Второй параметр функции `format-date` – строка-шаблон, которая окружает символы форматирования, заключенные в квадратные скобки (например, `[D01]`). Определено много разных символов форматирования, полный перечень см. в документе *http://www.w3.org/TR/xslt20/#function-format-date*.

Обсуждение

XSLT 1.0

Эти шаблоны годятся, если ваше приложение предназначено только для русскоговорящей аудитории. Но их можно и обобщить, организовав таблицу:

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="http://www.ora.com/XSLTCookbook/NS/dates">

<!-- США : us -->
<ckbk:month country="us" m="1"  name="January" abbrev="Jan" />
<ckbk:month country="us" m="2"  name="February" abbrev="Feb"/>
<ckbk:month country="us" m="3"  name="March" abbrev="Mar"/>
<ckbk:month country="us" m="4"  name="April" abbrev="Apr"/>
<ckbk:month country="us" m="5"  name="May" abbrev="May"/>
<ckbk:month country="us" m="6"  name="June" abbrev="Jun"/>
```

```
<ckbk:month country="us" m="7"  name="July" abbrev="Jul"/>
<ckbk:month country="us" m="8"  name="August" abbrev="Aug"/>
<ckbk:month country="us" m="9"  name="September" abbrev="Sep"/>
<ckbk:month country="us" m="10" name="October" abbrev="Oct"/>
<ckbk:month country="us" m="11" name="November" abbrev="Nov"/>
<ckbk:month country="us" m="12" name="December" abbrev="Dec"/>

<!-- Германия : de -->
<ckbk:month country="de" m="1"  name="Januar" abbrev="Jan"/>
<ckbk:month country="de" m="2"  name="Februar" abbrev="Feb"/>
<ckbk:month country="de" m="3"  name="Мдrz" abbrev="Мдr"/>
<ckbk:month country="de" m="4"  name="April" abbrev="Apr"/>
<ckbk:month country="de" m="5"  name="Mai" abbrev="Mai"/>
<ckbk:month country="de" m="6"  name="Juni" abbrev="Jun"/>
<ckbk:month country="de" m="7"  name="Juli" abbrev="Jul"/>
<ckbk:month country="de" m="8"  name="August" abbrev="Aug"/>
<ckbk:month country="de" m="9"  name="September" abbrev="Sep"/>
<ckbk:month country="de" m="10" name="Oktober" abbrev="Okt"/>
<ckbk:month country="de" m="11" name="November" abbrev="Nov"/>
<ckbk:month country="de" m="12" name="Dezember" abbrev="Dez"/>
<!-- Идея понятна ... -->

<!-- Для упрощения доступа сохраним элемент в переменной -->
<xsl:variable name="ckbk:months" select="document('')/*/ckbk:month"/>

</xsl:stylesheet>

<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="http://www.ora.com/XSLTCookbook/dates">

<xsl:include href="date-conversion.xsl"/>

<xsl:template name="ckbk:get-month-name">
  <xsl:param name="month"/>
  <xsl:param name="country" select="'us'"/>

  <xsl:value-of select="$ckbk:months[@country=$country and
       @m=$month]/@name"/>
</xsl:template>
```

XSLT 2.0

Функции `format-date()`, `format-time()`, `format-dateTime()` пригодны для решения многих задач. Помимо того, что продемонстрировано в этих рецептах,

они могут различными способами форматировать все компоненты даты и времени. Кроме того, можно задавать коды страны, календаря и языка, так что интернационализацию вы получаете задаром (в пределах имеющейся реализации).

См. также

Полную спецификацию функций форматирования дат в XSLT 2.0 см. в рецепте 4.10.

4.4. Вычисление юлианского и абсолютного дня, соответствующих заданной дате

Задача

Зная дату, требуется найти соответствующий ей юлианский или абсолютный номер дня.

ЮЛИАНСКИЙ ДЕНЬ И ДАТА ПО ЮЛИАНСКОМУ КАЛЕНДАРЮ

Не путайте юлианский день с юлианским календарем. Джозеф Дж. Скалигер (Joseph J. Scaliger) изобрел юлианский период, так чтобы каждому году можно было сопоставить целое положительное число, не заботясь о том, по какую сторону от начала новой эры он находится. Период начинается 1 января 4713 года до н.э. и длится 7980 лет. В астрономии юлианский период применяется для того, чтобы ассоциировать с каждым днем, начиная с 1 января 4713 года до н.э. уникальное число, которое и называется юлианским днем. Эту главу я начал писать в юлианский день с номером 2 452 376. Любознательный читатель может воспользоваться приведенным рецептом, чтобы узнать соответствующую ему дату. Другую схему абсолютной нумерации дней применили Н. Дершовиц и Э. Рейнгольд в своих календарных алгоритмах. В ней точкой отсчета является 1 января 1 года н.э. Номер дня в их системе я называю *абсолютным номером дня*. Абсолютный день 1 соответствует юлианскому дню 1 721 426.

Решение

XSLT 1.0

Следующий шаблон возвращает номер юлианского дня, если заданы год, месяц и день.

```
<xsl:template name="date:calculate-julian-day">
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>

  <xsl:variable name="a" select="floor((14 - $month) div 12)"/>
```

```
  <xsl:variable name="y" select="$year + 4800 - $a"/>
  <xsl:variable name="m" select="$month + 12 * $a - 3"/>

  <xsl:value-of select="$day + floor((153 * $m + 2) div 5) + $y * 365 +
       floor($y div 4) - floor($y div 100) + floor($y div 400) -
       32045"/>

</xsl:template>
```

Зная, как вычислить номер юлианского дня, легко написать шаблон для вычисления количества дней между любыми двумя датами:

```
<xsl:template name="ckbk:date-difference">
  <xsl:param name="from-year"/>
  <xsl:param name="from-month"/>
  <xsl:param name="from-day"/>
  <xsl:param name="to-year"/>
  <xsl:param name="to-month"/>
  <xsl:param name="to-day"/>

  <xsl:variable name="jd1">
    <xsl:call-template name="ckbk:calculate-julian-day">
      <xsl:with-param name="year" select="$from-year"/>
        <xsl:with-param name="month" select="$from-month"/>
        <xsl:with-param name="day" select="$from-day"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:variable name="jd2">
     <xsl:call-template name="ckbk:calculate-julian-day">
       <xsl:with-param name="year" select="$to-year"/>
         <xsl:with-param name="month" select="$to-month"/>
         <xsl:with-param name="day" select="$to-day"/>
     </xsl:call-template>
  </xsl:variable>
  <xsl:value-of select="$jd1 - $jd2"/>
</xsl:template>
```

Показанные ниже шаблоны преобразуют юлианский день в григорианскую дату в формате ГГГГ/ММ/ДД. Чтобы привести эту дату к форме, соответствующей конкретной локали, или выделить из нее отдельные компоненты, воспользуйтесь функциями `substring-before`, `substring-after` и `translate`.

```
<xsl:template name="date:julian-day-to-julian-date">
  <xsl:param name="j-day"/>
```

```
  <xsl:call-template name="date:julian-or-gregorian-date-elem">
    <xsl:with-param name="b" select="0"/>
    <xsl:with-param name="c" select="$j-day + 32082"/>
  </xsl:call-template>

</xsl:template>

<xsl:template name="date:julian-day-to-gregorian-date">
    <xsl:param name="j-day"/>

    <xsl:variable name="a" select="$j-day + 32044"/>
    <xsl:variable name="b" select="floor((4 * $a + 3) div 146097)"/>
    <xsl:variable name="c" select="$a - 146097 * floor($b div 4)"/>

    <xsl:call-template name="date:julian-or-gregorian-date-elem">
           <xsl:with-param name="b" select="$b"/>
           <xsl:with-param name="c" select="$c"/>
    </xsl:call-template>

</xsl:template>

<!-- Вспомогательный шаблон, используемый и для григорианского, и для
юлианского календаря -->
<xsl:template name="date:julian-or-gregorian-date-elem">
    <xsl:param name="b"/>
    <xsl:param name="c"/>

    <xsl:variable name="d" select="floor((4 * $c + 3) div 1461)"/>
    <xsl:variable name="e" select="$c - floor((1461 * $d) div 4)"/>
    <xsl:variable name="m" select="floor((5 * $e + 2) div 153)"/>

    <xsl:variable name="day"
          select="$e - floor((153 * $m + 2) div 5) + 1"/>

    <xsl:variable name="month"
          select="$m + 3 - (12 * floor($m div 10))"/>

    <xsl:variable name="year"
          select="100 * $b + $d - 4800 + floor($m div 10)"/>

    <xsl:value-of select="concat($year,'/',$month,'/',$day)"/>

</xsl:template>
```

С помощью следующих шаблонов нетрудно выполнить преобразование юлианского дня в абсолютный и наоборот:

```
<xsl:template name="ckbk:julian-day-to-absolute-day">
    <xsl:with-param name="j-day">
    <xsl:value-of select="j-day - 1721425"/>
 </xsl:template>


<xsl:template name="ckbk:julian-day-to-julian-day">
    <xsl:param name="abs-day">
    <xsl:value-of select="$abs-day + 1721425"/>
 </xsl:template>
```

Теперь можно выразить преобразование между абсолютным днем и григорианским календарем в терминах уже имеющегося преобразования между юлианским днем и григорианским календарем.

```
<xsl:template name="ckbk:date-to-absolute-day">
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>


  <xsl:call-template name="ckbk:julian-day-to-absolute-day">
    <xsl:with-param name="j-day">
       <xsl:call-template name="ckbk:date-to-julian-day">
         <xsl:with-param name="year" select="$year"/>
         <xsl:with-param name="month" select="$month"/>
         <xsl:with-param name="day" select="$day"/>
       </xsl:call-template>
     </xsl:with-param>
   </xsl:call-template>
</xsl:template>


<xsl:template name="ckbk:absolute-day-to-date">
  <xsl:param name="abs-day"/>


  <xsl:call-template name="ckbk:julian-day-to-date">
       <xsl:with-param name="j-day">
           <xsl:call-template name="ckbk:absolute-day-to-julian-day">
                 <xsl:with-param name="abs-day" select="$abs-day"/>
           </xsl:call-template>
       </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

XSLT 2.0

В версии 2.0 необходимость в функциях вычисления юлианского и абсолютного дня не так насущна, поскольку поддержка операций с датами встроена в сам язык. Например:

```
<dateDiff>
<xsl:value-of select="xs:date('2005-02-21') - xs:date('2005-01-01')"/>
</dateDiff>
```

возвращает

```
<dateDiff>P51D</dateDiff>
```

Однако в некоторых приложениях эти функции могут понадобиться для других целей. Поэтому преобразуем рассмотренные выше шаблоны в функции, принимающие параметры типа `xs:date` вместо отдельных компонентов:

```
<xsl:function name="ckbk:calculate-julian-day" as="xs:integer">
  <xsl:param name="date" as="xs:date"/>

  <xsl:variable name="year" select="year-from-date($date)"  as="xs:integer"/>
  <xsl:variable name="month" select="month-from-date($date)" as="xs:integer"/>
  <xsl:variable name="day" select="month-from-date($date)" as="xs:integer"/>

  <xsl:variable name="a" select="(14 - $month) idiv 12" as="xs:integer"/>
  <xsl:variable name="y" select="$year + 4800 - $a" as="xs:integer"/>
  <xsl:variable name="m" select="$month + 12 * $a - 3" as="xs:integer"/>

  <xsl:sequence select="$day + ((153 * $m + 2) idiv 5) + $y * 365 +
       floor($y div 4) - ($y idiv 100) + ($y idiv 400) - 32045"/>

</xsl:function>
```

Обсуждение

Юлианский и абсолютный день полезны, потому что существенно упрощают другие алгоритмы работы с датами. В этой главе мы часто будем прибегать к таким преобразованиям. Подобные схемы нумерации играют роль единой валюты для всех рассматриваемых календарных систем. Если вам потребуется перевести дату из еврейского календаря в мусульманский, достаточно будет выполнить промежуточное приведение к абсолютному дню.

4.5. Вычисление номера недели, соответствующего заданной дате

Задача

Требуется найти номер недели года, в которую попадает заданная дата.

Решение

XSLT 1.0

Номер недели изменяется в диапазоне от 1 до 53. В большинстве годов количество недель равно 52, но в тех, которые содержат 53 четверга, недель тоже 53.

Для решения воспользуемся шаблоном вычисления юлианского дня:

```
<xsl:template name="ckbk:calculate-week-number">
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>

  <xsl:variable name="j-day">
    <xsl:call-template name="ckbk:calculate-julian-day">
      <xsl:with-param name="year" select="$year"/>
      <xsl:with-param name="month" select="$month"/>
      <xsl:with-param name="day" select="$day"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:variable name="d4"
       select="($j-day + 31741 - ($j-day mod 7))
           mod 146097 mod 36524 mod 1461"/>
  <xsl:variable name="L" select="floor($d4 div 1460)"/>

  <xsl:variable name="d1" select="(($d4 - $L) mod 365) + $L"/>

  <xsl:value-of select="floor($d1 div 7) + 1"/>

</xsl:template>
```

В этой функции предполагается, что неделя начинается с понедельника. В большинстве же функций в этой главе используется более распространенное соглашение о том, что первый день недели – воскресенье. Объяснение этой странности см. в рецептах, относящихся к календарю ISO.

XSLT 2.0

Воспользуемся функцией `format-date()`:

```
<xsl:function name="ckbk:calculate-week-number" as="xs:integer">
  <xsl:param name="date" as="xs:date"/>
  <xsl:sequence select="xs:integer(format-date($date,'[W]'))"/>
</xsl:function>
```

Обсуждение

Номер недели – это число, сопоставляемое каждой неделе года. Первой в году считается неделя, содержащая 4 января, или, что эквивалентно, неделя, в которую попадает первый четверг января. Неделя, которая начинается в одном году, а заканчивается в следующем, считается относящейся к тому году, в который попадает больше дней. Такое происходит, когда год начинается с четверга или, если он високосный, со среды. В США это соглашение в настоящее время не применяется.

См. также

См. рецепт 4.8 выше в этой главе.

4.6. Юлианский календарь

Задача

Требуется работать в старой юлианской системе летоисчисления.

Решение

```
<xsl:template name="ckbk:julian-date-to-julian-day">
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>

  <xsl:variable name="a" select="floor((14 - $month) div 12)"/>
  <xsl:variable name="y" select="$year + 4800 - $a"/>
  <xsl:variable name="m" select="$month + 12 * $a - 3"/>

  <xsl:value-of
      select="$day + floor((153 * $m + 2) div 5) + 365 * $y
      + floor($y div 4) - 32083"/>

</xsl:template>
```

Зная юлианский день, вы можете воспользоваться приведенными в этой главе рецептами для форматирования дат, выполнения математических операций над ними и преобразования в другие календарные системы.

Обсуждение

Юлианский календарь сегодня применяется редко (исключение составляет Русская православная церковь). Он был заменен григорианским из-за неточного предположения о том, что год состоит из 365 1/4 дней. На самом деле, средняя длина когда составляет 365.2425 дней, поэтому с течением времени в юлианском календаре времена года смещаются.

4.7. Календарь ISO

Задача

Требуется работать с датами, представленными в международном стандарте ISO-8601[1].

Читатели, знакомые со спецификацией XML Schema, не должны путать календарь ISO с форматом ISO, применяемым для записи григорианских дат (например, 2002-04-12 или 2002-0412T09:26:00). Стандарт 8601 описывает и календарь, и способ форматирования. Данный рецепт относится к календарю ISO.

Решение

Для работы с ISO-датами (а также для вычисления дат некоторых праздников) понадобится функция, вычисляющая абсолютный день k-ого дня недели, предшествующего заданному абсолютному дню или следующего за ним. Например, первый понедельник (k=1) не позднее 4 января 2004 года (абсолютный день 731 584) приходится на 29 декабря 2003 года (абсолютный день 731 578).

```
<xsl:template name="ckbk:k-date-on-or-before-abs-day">
     <xsl:param name="abs-day"/>
     <xsl:param-name="k"/>
    <xsl:value-of select="$abs-day - (($abs-day - $k) mod 7)"/>
</xsl:template>
```

Теперь для преобразования ISO-даты в абсолютный день нужно лишь вычислить количество абсолютных дней в предшествующих годах и прибавить количество дней в самой ISO-дате:

```
<xsl:template name="ckbk:iso-date-to-absolute-day">
    <xsl:param name="iso-week"/>
    <xsl:param name="iso-day"/>
    <xsl:param name="iso-year"/>

    <xsl:variable name="a">
           <xsl:call-template name="ckbk:date-to-absolute-day">
              <xsl:with-param name="year" select="$iso-year"/>
              <xsl:with-param name="month" select="1"/>
              <xsl:with-param name="day" select="4"/>
           </xsl:call-template>
    </xsl:variable>
```

[1] ISO – это Международная организация по стандартизации (International Organization for Standardization). Аббревиатура ISO не только проще произносится, но еще и ассоциируется с греческим словом *isos*, означающим *равный*.

```
    <xsl:variable name="days-in-prior-yrs">
           <xsl:call-template name="ckbk:k-day-on-or-before-abs-day">
              <xsl:with-param name="abs-day" select="$a"/>
              <xsl:with-param name="k" select="1"/>
           </xsl:call-template>
    </xsl:variable>

    <xsl:variable name="days-in-prior-weeks-this-yr"
           select="7 * ($iso-week - 1)"/>

    <xsl:variable name="prior-days-this-week" select="$iso-day - 1"/>

    <xsl:value-of select="$days-in-prior-yrs +
          $days-in-prior-weeks-this-yr + $prior-days-this-week"/>
</xsl:template>
```

Для преобразования абсолютного дня в ISO-дату мы сначала пытаемся определить год, предположив, что он такой же, как для григорианской даты минус 3 дня. Это предположение неверно только, если абсолютный день приходится на дату между 1 и 3 января следующего года. Чтобы исправить возможную ошибку, мы сравниваем дату с 1 января следующего года с помощью уже имеющегося шаблона `iso-date-to-absolute-day`. Окончательно выяснив, чему равен ISO-год, мы находим номер недели и день, вычисляя смещение от 1 января этого года. ISO-дата возвращается в формате *год-месяц-день*. Такой формат определен в стандарте ISO, чтобы не спутать с датой по григорианскому календарю.

```
<xsl:template name="ckbk:absolute-day-to-iso-date">
  <xsl:param name="abs-day"/>

  <xsl:variable name="d">
    <xsl:call-template name="ckbk:absolute-day-to-date">
      <xsl:with-param name="abs-day" select="$abs-day - 3"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:variable name="approx" select="substring-before($d,'/')"/>

  <xsl:variable name="iso-year">
    <xsl:variable name="a">
      <xsl:call-template name="ckbk:iso-date-to-absolute-day">
           <xsl:with-param name="iso-week" select="1"/>
             <xsl:with-param name="iso-day" select="1"/>
           <xsl:with-param name="iso-year" select="$approx + 1"/>
          </xsl:call-template>
    </xsl:variable>
    <xsl:choose>
```

```
          <xsl:when test="$abs-day >= $a">
            <xsl:value-of select="$approx + 1"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$approx"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>

      <xsl:variable name="iso-week">
            <xsl:variable name="a">
              <xsl:call-template name="ckbk:iso-date-to-absolute-day">
                  <xsl:with-param name="iso-week" select="1"/>
                  <xsl:with-param name="iso-day" select="1"/>
                  <xsl:with-param name="iso-year" select="$iso-year"/>
              </xsl:call-template>
            </xsl:variable>
            <xsl:value-of select="1 + floor(($abs-day - $a) div 7)"/>
    </xsl:variable>

    <xsl:variable name="iso-day">
       <xsl:variable name="a" select="$abs-day mod 7"/>
      <xsl:choose>
          <xsl:when test="not($a)">
               <xsl:value-of select="7"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$a"/>
          </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <xsl:value-of select="concat($iso-year,'-W',$iso-week,'-',$iso-day)"/>

  </xsl:template>
```

Обсуждение

В европейских приложениях для торговых и промышленных предприятий часто бывает необходимо знать номер недели в году. В календаре ISO дата задается в виде года по григорианскому календарю, номера недели (1–53) и номера дня недели (1–7, причем, согласно стандарту ISO, первым днем недели считается понедельник). Неделя, пересекающая границу двух годов, считается принадлежащей тому году, в который попадает наибольшее число дней. Согласно этому правилу, первая неделя по календарю ISO может начинаться как 4 января, так и 29 декабря предыдущего года. Аналогично последняя неделя года по календарю

ISO может заканчиваться и 28 декабря, и 3 января следующего года. Например, первая ISO-неделя 2004 года началась 29 декабря 2003 года![1] Следовательно, чтобы определить, когда начинается ISO-неделя, необходимо найти последний понедельник, приходящийся на 4 января или ранее.

См. также

Увидеть календарь ISO в действии можно на странице *http://personal.ecu.edu/mccartyr/isowdcal.html*.

4.8. Исламский календарь

Задача

Требуется работать с датами в исламской системе летоисчисления.

Трудно, если вообще невозможно, придумать универсально пригодные алгоритмы для исламского календаря. Связано это с тем, что месяц начинается в тот день, когда человек впервые видит серп луны. Вычислить даты новолуния можно абсолютно точно, но вот видимость луны зависит от таких факторов, как погода и положение наблюдателя. Поэтому очень сложно заранее сказать, когда начнется новый месяц. Хуже того, некоторым мусульманам достаточно, чтобы луну увидел местный наблюдатель, тогда как другие требуют, чтобы наблюдение было произведено двумя заслуживающими доверия мусульманами. Исключением является Саудовская Аравия, где в основу календаря положены астрономические вычисления, а не визуальные наблюдения. Приведенные ниже алгоритмы работы с исламским календарем могут давать ошибку в несколько дней.

Решение

Последний день месяца по исламскому календарю можно вычислить точно, если считать, что в нечетных месяцах 30 дней, а в четных - 29, за исключением високосных лет:

```
<xsl:template name="date:last-day-of-islamic-month">
  <xsl:param name="month"/>
  <xsl:param name="year"/>

  <xsl:variable name="islamic-leap-year"
        select="(11 * $year + 14) mod 30 &lt; 11"/>

  <xsl:choose>
    <xsl:when test="$month mod 2 or ($month = 12 and $islamic-leap-year)">
      <xsl:value-of select="30"/>
```

[1] А вы думали, что стандарты призваны упростить жизнь?

```
    </xsl:when>
    <xsl:otherwise>
       <xsl:value-of select="29"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Летоисчисление по исламскому календарю начинается с Хиджры – даты переселения пророка Мухаммада в Медину. Для большинства мусульман этот момент совпадает с закатом солнца 15 июля 622 года н.э. (по юлианскому календарю). Этой дате соответствует абсолютный день 227 015, отсюда и слагаемое 227 014 в формуле вычисления абсолютного дня по дате исламского календаря:

```
<xsl:template name="date:islamic-date-to-absolute-day">
  <xsl:param name="year"/>
  <xsl:param name="month"/>
  <xsl:param name="day"/>

  <xsl:value-of select="$day + 29 * ($month - 1) + floor($month div 2) + 354
       * ($year - 1) + floor((11 * $year + 3) div 30) + 227014"/>
</xsl:template>
```

Преобразование абсолютного дня в исламскую дату выполняется не так, как в оригинальном Lisp-коде. Я воспользовался математикой с плавающей точкой, чтобы избежать поиска, присутствующего в реализации на Lisp. Авторы Lisp-кода стремились ограничиться вычислениями с 24-разрядными целыми числами, сохранив при этом максимальную точность. Однако примененные ими методы не переносятся на XSLT. Учитывая, что в XSLT 1.0 все равно используются только числа с плавающей точкой, не имеет смысла всеми силами пытаться избежать вычислений с ними. Встречающиеся в коде константы происходят от средней продолжительности лунного месяца, которая равна 29.530555... дням. Номер месяца вычисляется приближенно, а затем корректируется, если в результате получается, что значения дня меньше 1. Установив, чему равны год и месяц, день можно вычислить как смещение от начала года:

```
<xsl:template name="date:absolute-day-to-islamic-date">
  <xsl:param name="abs-day"/>

  <xsl:variable name="year"
        select="floor(($abs-day - 227014) div 354.36667) + 1"/>

  <xsl:variable name="month">
    <xsl:variable name="a"
       select="$abs-day - 227014 - floor((11 * $year + 3) div 30) -
               354 * ($year - 1)"/>
    <xsl:variable name="approx" select="floor($a div 29.53056)+1"/>
    <xsl:choose>
       <xsl:when test="(29 * ($approx - 1) + floor($approx div 2)) -
```

```
          $a &lt; 1">
            <xsl:value-of select="$approx - 1"/>
          </xsl:when>
          <xsl:otherwise>
             <xsl:value-of select="$approx"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>

      <xsl:variable name="day">
        <xsl:variable name="a">
          <xsl:call-template name="date:islamic-date-to-absolute-day">
              <xsl:with-param name="year" select="$year"/>
              <xsl:with-param name="month" select="$month"/>
              <xsl:with-param name="day" select="1"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$abs-day - $a + 1"/>
      </xsl:variable>

      <xsl:value-of select="concat($year,'/',$month,'/',$day)"/>

    </xsl:template>
```

Обсуждение

Календарь Хиджры или исламский календарь интересен тем, что основан только на лунных циклах, поэтому исламские месяцы попадают в разные времена года. В исламском годе приблизительно 354.36 дней. Первый день исламского календаря обозначается 1 A. H. (After Hijra – после Хиджры) – дата переселения Мухаммада из Мекки в Медину. Исламский календарь имеет глубокий религиозный смысл для правоверных мусульман и почти всегда основывается на визуальных наблюдениях за луной. Календарь на будущее можно приблизительно рассчитать, но пользоваться им следует только для чернового планирования.

См. также

Ближе познакомиться с мусульманским календарем можно на странице *http://www.sufisattari.com/calendar.html*.

Соглашения, принятые в различных мусульманских странах, описаны на странице *http://www.math.nus.edu.sg/aslaksen/calendar/islamic.shtml*.

4.9. Еврейский календарь

Задача

Требуется работать с датами в еврейской системе летоисчисления.

Решение

Для эффективной работы с еврейским календарем понадобятся несколько вспомогательных шаблонов. В еврейском календаре обычный год состоит из 12 месяцев, а високосный – из 13. Високосными считаются 3, 6, 8, 11, 14, 17 и 19 года *метонова цикла* (см. ниже раздел «Обсуждение»). Точная формулировка условия високосности такова: $7y + 1 \bmod 19 < 7$. Это позволяет написать шаблон для определения последнего месяца года по еврейскому календарю:

```
<xsl:template name="ckbk:last-month-of-hebrew-year">
  <xsl:param name="year"/>
  <xsl:choose>
    <xsl:when test="(7 * $year + 1) mod 19 &lt; 7">
          <xsl:value-of select="13"/>
    </xsl:when>
    <xsl:otherwise>
          <xsl:value-of select="12"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Для вычисления количества дней в заданном месяце или годе нужно предварительно инкапсулировать сложные правила определения начала года в еврейском календаре. Подробные объяснения см. в работе Дершовица и Рейнгольда.

```
<!-- Количество дней, прошедших с воскресенья, предшествующего началу
еврейского календаря, до усредненной даты начала месяца Тишрей -->

<xsl:template name="ckbk:hebrew-calendar-elapsed-days">
  <xsl:param name="year"/>

  <xsl:variable name="hebrew-leap-year"
    select="(7 * $year + 1) mod 19 &lt; 7"/>
  <xsl:variable name="hebrew-leap-year-last-year"
    select="(7 * ($year - 1) + 1) mod 19 &lt; 7"/>

  <xsl:variable name="months-ellapsed"
     select="235 * floor(($year -1) div 19) +
       12 * (($year -1) mod 19) +
       floor((7 * (($year - 1) mod 19) + 1) div 19)"/>

  <xsl:variable name="parts-ellapsed"
    select="13753 * $months-ellapsed + 5604"/>

  <xsl:variable name="day" select="1 + 29 * $months-ellapsed +
                  floor($parts-ellapsed div 25920)"/>
```

```
    <xsl:variable name="parts" select="$parts-ellapsed mod 25920"/>

    <xsl:variable name="alternative-day">
      <xsl:choose>
         <xsl:when test="$parts >= 19440">
              <xsl:value-of select="$day + 1"/>
         </xsl:when>
         <xsl:when test="$day mod 7 = 2 and $parts >= 9924 and
              not($hebrew-leap-year)">
           <xsl:value-of select="$day + 1"/>
         </xsl:when>
         <xsl:when test="$day mod 7 = 1 and $parts >= 16789 and
              $hebrew-leap-year-last-year">
            <xsl:value-of select="$day + 1"/>
         </xsl:when>
         <xsl:otherwise>
              <xsl:value-of select="$day"/>
         </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="$alternative-day mod 7 = 0">
         <xsl:value-of select="$alternative-day + 1"/>
      </xsl:when>
      <xsl:when test="$alternative-day mod 7 =3">
         <xsl:value-of select="$alternative-day + 1"/>
      </xsl:when>
      <xsl:when test="$alternative-day mod 7 = 5">
         <xsl:value-of select="$alternative-day + 1"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$alternative-day"/>
      </xsl:otherwise>
    </xsl:choose>

  </xsl:template>
```

Число дней в году вычисляется как разность между количеством дней до начала последующего и предыдущего года:

```
  <xsl:template name="ckbk:days-in-hebrew-year">
    <xsl:param name="year"/>

    <xsl:variable name="e1">
```

```
    <xsl:call-template name="ckbk:hebrew-calendar-ellapsed-days">
       <xsl:with-param name="year" select="$year + 1"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:variable name="e2">
    <xsl:call-template name="ckbk:hebrew-calendar-ellapsed-days">
       <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:value-of select="$e1 - $e2"/>
</xsl:template>
```

Хешван и Кислев – восьмой и девятый месяцы еврейского года, число дней в них переменно. Необходимо знать, когда Хешван длинный, а Кислев короткий, поэтому введем два предиката:

```
<xsl:template name="date:long-heshvan">
      <xsl:param name="year"/>

  <xsl:variable name="days">
       <xsl:call-template name="date:days-in-hebrew-year">
          <xsl:with-param name="year" select="$year"/>
       </xsl:call-template>
  </xsl:variable>

      <xsl:if test="$days mod 10 = 5">
           <xsl:value-of select="true()"/>
      </xsl:if>
</xsl:template>

<xsl:template name="date:short-kislev">
      <xsl:param name="year"/>

      <xsl:variable name="days">
           <xsl:call-template name="date:days-in-hebrew-year">
              <xsl:with-param name="year" select="$year"/>
           </xsl:call-template>
      </xsl:variable>

      <xsl:if test="$days mod 10 = 3">
            <xsl:value-of select="true()"/>
      </xsl:if>
</xsl:template>
```

При кодировании шаблонов-предикатов на XSLT 1.0 нужно, чтобы они возвращали `true()` (или `'true'`) в качестве истинного значения, но `''` (null string) – в качестве ложного. Проблема в том, что шаблоны возвращают деревья, а вычисление любого дерева, пусть даже всего из одного узла, который содержит `false()` или `''`, дает true. Но у дерева, содержащего узел `''`, есть одно достоинство: оно эффективно вычисляется как булевское значение с помощью преобразования функцией `string()`. Это одна из многих странностей XSLT. В XSLT 2.0 можно вместо этого просто использовать функции, возвращающие булевские значения.

Теперь почти все готово к написанию стандартных функций для работы с датой и временем, которые приводились в других рецептах. Первая из них дает последний день заданного месяца и года еврейского календаря:

```
<xsl:template name="ckbk:last-day-of-hebrew-month">
     <xsl:param name="month"/>
     <xsl:param name="year"/>

     <xsl:variable name="hebrew-leap-year"
          select="(7 * $year + 1) mod 19 &lt; 7"/>

    <xsl:variable name="long-heshvan">
      <xsl:call-template name="ckbk:long-heshvan">
        <xsl:with-param name="year" select="$year"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:variable name="short-kislev">
      <xsl:call-template name="ckbk:short-kislev">
        <xsl:with-param name="year" select="$year"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:choose>
      <xsl:when test="$month=12 and $hebrew-leap-year">
        <xsl:value-of select="30"/>
      </xsl:when>
      <xsl:when test="$month=8 and string($long-heshvan)">
        <xsl:value-of select="30"/>
      </xsl:when>
      <xsl:when test="$month=9 and string($short-kislev)">
        <xsl:value-of select="29"/>
      </xsl:when>
      <xsl:when test="$month=13">
        <xsl:value-of select="29"/>
```

```
      </xsl:when>
      <xsl:when test="$month mod 2 = 0">
        <xsl:value-of select="29"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="30"/>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Следующий рекурсивный шаблон суммирует последние дни в заданном диапазоне месяцев заданного года. Он используется при преобразовании даты еврейского календаря в абсолютную.

```
<xsl:template name="date:sum-last-day-in-hebrew-months">
    <xsl:param name="year"/>
    <xsl:param name="from-month"/>
    <xsl:param name="to-month"/>
    <xsl:param name="accum" select="0"/>

    <xsl:choose>
       <xsl:when test="$from-month &lt;= $to-month">
          <xsl:call-template name="date:sum-last-day-in-hebrew-months">
           <xsl:with-param name="year" select="$year"/>
           <xsl:with-param name="from-month" select="$from-month+1"/>
           <xsl:with-param name="to-month" select="$to-month"/>
           <xsl:with-param name="accum">
             <xsl:variable name="temp">
                <xsl:call-template name="date:last-day-of-hebrew-month">
                  <xsl:with-param name="year" select="$year"/>
                  <xsl:with-param name="month" select="$from-month"/>
                </xsl:call-template>
             </xsl:variable>
             <xsl:value-of select="$temp + $accum"/>
           </xsl:with-param>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
           <xsl:value-of select="$accum"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template name="date:hebrew-date-to-absolute-day">
    <xsl:param name="year"/>
```

```
<xsl:param name="month"/>
<xsl:param name="day"/>

<xsl:variable name="prior-months-days">
   <xsl:choose>
     <xsl:when test="7 > $month"> <!-- before Tishri -->
       <xsl:variable name="last-month-of-year">
          <xsl:call-template name="date:last-month-of-hebrew-year">
             <xsl:with-param name="year" select="$year"/>
          </xsl:call-template>
       </xsl:variable>
       <!-- Добавить дни до и после месяца Нисан -->
       <xsl:variable name="days-before-nisan">
          <xsl:call-template name="date:sum-last-day-in-hebrew-months">
             <xsl:with-param name="year" select="$year"/>
             <xsl:with-param name="from-month" select="7"/>
             <xsl:with-param name="to-month"
              select="$last-month-of-year"/>
             </xsl:call-template>
           </xsl:variable>
           <xsl:call-template name="date:sum-last-day-in-hebrew-months">
              <xsl:with-param name="year" select="$year"/>
              <xsl:with-param name="from-month" select="1"/>
              <xsl:with-param name="to-month" select="$month - 1"/>
              <xsl:with-param name="accum" select="$days-before-nisan"/>
           </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <!-- количество дней в предшествующих месяцах этого года -->
          <xsl:call-template name="date:sum-last-day-in-hebrew-months">
             <xsl:with-param name="year" select="$year"/>
             <xsl:with-param name="from-month" select="7"/>
             <xsl:with-param name="to-month" select="$month - 1"/>
          </xsl:call-template>
       </xsl:otherwise>
   </xsl:choose>
</xsl:variable>

<xsl:variable name="days-in-prior-years">
   <xsl:call-template name="date:hebrew-calendar-ellapsed-days">
       <xsl:with-param name="year" select="$year"/>
   </xsl:call-template>
</xsl:variable>

<!-- 1373429 дней до абсолютного дня 1 -->
```

```
    <xsl:value-of select="$day + $prior-months-days +
           $days-in-prior-years - 1373429"/>
</xsl:template>
```

Прежде чем приступать к реализации шаблона `absolute-days-to-hebrew-date`, нам понадобятся еще две рекурсивные утилиты суммирования, которые помогут вычислить истинный год и месяц, соответствующие абсолютному дню, по их приблизительным значениям:

```
<xsl:template name="ckbk:fixup-hebrew-year">
    <xsl:param name="start-year"/>
    <xsl:param name="abs-day"/>

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

    <xsl:variable name="next">
      <xsl:call-template name="ckbk:hebrew-date-to-absolute-day">
         <xsl:with-param name="month" select="7"/>
         <xsl:with-param name="day" select="1"/>
         <xsl:with-param name="year" select="$start-year + 1"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:choose>
       <xsl:when test="$abs-day >= $next">
          <xsl:call-template name="ckbk:fixup-hebrew-year">
           <xsl:with-param name="start-year" select="$start-year+1"/>
           <xsl:with-param name="abs-day" select="$abs-day"/>
           <xsl:with-param name="accum" select="$accum + 1"/>
          </xsl:call-template>
       </xsl:when>
       <xsl:otherwise>
          <xsl:value-of select="$accum"/>
       </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template name="ckbk:fixup-hebrew-month">
    <xsl:param name="year"/>
    <xsl:param name="start-month"/>
    <xsl:param name="abs-day"/>

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

    <xsl:variable name="next">
```

```
          <xsl:call-template name="ckbk:hebrew-date-to-absolute-day">
            <xsl:with-param name="month" select="$start-month"/>
            <xsl:with-param name="day">
             <xsl:call-template name="ckbk:last-day-of-hebrew-month">
                <xsl:with-param name="month" select="$start-month"/>
                <xsl:with-param name="year" select="$year"/>
             </xsl:call-template>
            </xsl:with-param>
            <xsl:with-param name="year" select="$year"/>
          </xsl:call-template>
      </xsl:variable>

      <xsl:choose>
         <xsl:when test="$abs-day > $next">
            <xsl:call-template name="ckbk:fixup-hebrew-month">
             <xsl:with-param name="year" select="$year"/>
             <xsl:with-param name="start-month" select="$start-month + 1"/>
             <xsl:with-param name="abs-day" select="$abs-day"/>
             <xsl:with-param name="accum" select="$accum + 1"/>
            </xsl:call-template>
         </xsl:when>
         <xsl:otherwise>
             <xsl:value-of select="$accum"/>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:template>

   <xsl:template name="date:absolute-day-to-hebrew-date">
       <xsl:param name="abs-day"/>

      <xsl:variable name="year"
         <xsl:variable name="approx"
             select="floor(($abs-day + 1373429) div 366)"/>
         <xsl:variable name="fixup">
            <xsl:call-template name="date:fixup-hebrew-year">
             <xsl:with-param name="start-year" select="$approx"/>
             <xsl:with-param name="abs-day" select="$abs-day"/>
            </xsl:call-template>
         </xsl:variable>
         <xsl:value-of select="$approx + $fixup"/>
      </xsl:variable>

      <xsl:variable name="month">
         <xsl:variable name="first-day-of-year">
```

```
        <xsl:call-template name="date:hebrew-date-to-absolute-day">
          <xsl:with-param name="month" select="1"/>
          <xsl:with-param name="day" select="1"/>
          <xsl:with-param name="year" select="$year"/>
        </xsl:call-template>
    </xsl:variable>

    <xsl:variable name="approx">
        <xsl:choose>
          <xsl:when test="$abs-day &lt; $first-day-of-year">
            <xsl:value-of select="7"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="1"/>
          </xsl:otherwise>
        </xsl:choose>
    </xsl:variable>

        <xsl:variable name="fixup">
          <xsl:call-template name="date:fixup-hebrew-month">
            <xsl:with-param name="year" select="$year"/>
            <xsl:with-param name="start-month" select="$approx"/>
            <xsl:with-param name="abs-day" select="$abs-day"/>
          </xsl:call-template>
        </xsl:variable>

        <xsl:value-of select="$approx + $fixup"/>
    </xsl:variable>

    <xsl:variable name="day">
        <xsl:variable name="days-to-first-of-month">
          <xsl:call-template name="date:hebrew-date-to-absolute-day">
            <xsl:with-param name="month" select="$month"/>
            <xsl:with-param name="day" select="1"/>
            <xsl:with-param name="year" select="$year"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$abs-day - ($days-to-first-of-month - 1)"/>
    </xsl:variable>

    <xsl:value-of select="concat($year,'-',$month,'-',$day)"/>

</xsl:template>
```

В предыдущем примере мы создали три похожих рекурсивных шаблона для вычисления суммы значений разных функций. Хорошо бы иметь обобщенную утилиту для суммирования значений произвольной функции. Читатели, знакомые с языком Lisp, наверное, догадались, что в оригинальном коде для этой цели применялся макрос. В главе 13 показано, как можно реализовать обобщенное программирование на языке XSLT и тем самым заметно упростить этот рецепт.

Обсуждение

Еврейский календарь – самый сложный из рассмотренных в этой главе. Потому и код для работы с ним получился таким сложным. Сложность еврейского календаря обусловлена тем, что месяцы в нем всегда лунные, но пасха (праздник Песах) обязательно должна приходиться на весну. Если в других календарях число месяцев в году фиксировано, то в еврейском обычный год состоит из 12 месяцев, а високосный – из 13.

4.10. Форматирование даты и времени

Задача

Требуется отформатировать дату и время в соответствии с заданной форматной строкой.

Решение

Мы повторно используем многие ранее разработанные в этой главе шаблоны. В шаблоне `format-date-time` спецификаторы формата имеют вид `%x` (см. ниже), а прочие символы выводятся буквально. По умолчанию подразумевается формат ISO для даты и времени по григорианскому календарю.

```
<xsl:template name="ckbk:format-date-time">
    <xsl:param name="year"/>
    <xsl:param name="month"/>
    <xsl:param name="day"/>
    <xsl:param name="hour"/>
    <xsl:param name="minute"/>
    <xsl:param name="second"/>
    <xsl:param name="time-zone"/>
    <xsl:param name="format" select="'%Y-%m-%dT%H:%M:%S%z'"/>

    <xsl:choose>
      <xsl:when test="contains($format, '%')">
        <xsl:value-of select="substring-before($format, '%')"/>
      </xsl:when>
```

```
  <xsl:otherwise>
    <xsl:value-of select="$format"/>
  </xsl:otherwise>
</xsl:choose>

<xsl:variable name="code"
              select="substring(substring-after($format, '%'), 1, 1)"/>
<xsl:choose>

  <!-- Сокращенное название дня недели -->
  <xsl:when test="$code='a'">
    <xsl:variable name="day-of-the-week">
      <xsl:call-template name="ckbk:calculate-day-of-the-week">
        <xsl:with-param name="year" select="$year"/>
        <xsl:with-param name="month" select="$month"/>
        <xsl:with-param name="day" select="$day"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:call-template name="ckbk:get-day-of-the-week-abbreviation">
      <xsl:with-param name="day-of-the-week"
         select="$day-of-the-week"/>
    </xsl:call-template>
  </xsl:when>

  <!-- Полное название дня недели -->
  <xsl:when test="$code='A'">
    <xsl:variable name="day-of-the-week">
      <xsl:call-template name="ckbk:calculate-day-of-the-week">
        <xsl:with-param name="year" select="$year"/>
        <xsl:with-param name="month" select="$month"/>
        <xsl:with-param name="day" select="$day"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:call-template name="ckbk:get-day-of-the-week-name">
      <xsl:with-param name="day-of-the-week"
                      select="$day-of-the-week"/>
    </xsl:call-template>
  </xsl:when>

  <!-- Сокращенное название месяца -->
  <xsl:when test="$code='b'">
    <xsl:call-template name="ckbk:get-month-abbreviation">
      <xsl:with-param name="month" select="$month"/>
    </xsl:call-template>
```

```
</xsl:when>

<!-- Полное название месяца -->
<xsl:when test="$code='B'">
  <xsl:call-template name="ckbk:get-month-name">
    <xsl:with-param name="month" select="$month"/>
  </xsl:call-template>
</xsl:when>

<!-- Представление даты и времени в текущей локали -->
<xsl:when test="$code='c'">
  <xsl:text>[не реализовано]</xsl:text>
</xsl:when>

<!-- День месяца в виде десятичного числа (01 - 31) -->
<xsl:when test="$code='d'">
  <xsl:value-of select="format-number($day,'00')"/>
</xsl:when>

<!-- Час в 24-часовом формате (00 - 23) -->
<xsl:when test="$code='H'">
  <xsl:value-of select="format-number($hour,'00')"/>
</xsl:when>

<!-- Час в 12-часовом формате (01 - 12) -->
<xsl:when test="$code='I'">
  <xsl:choose>
    <xsl:when test="$hour = 0">12</xsl:when>
    <xsl:when test="$hour &lt; 13">
      <xsl:value-of select="format-number($hour,'00')"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="format-number($hour - 12,'00')"/>
   </xsl:otherwise>
  </xsl:choose>
</xsl:when>

<!-- День года в виде десятичного числа (001 - 366) -->
<xsl:when test="$code='j'">
  <xsl:variable name="diff">
   <xsl:call-template name="ckbk:date-difference">
     <xsl:with-param name="from-year" select="$year"/>
     <xsl:with-param name="from-month" select="1"/>
     <xsl:with-param name="form-day" select="1"/>
```

```
      <xsl:with-param name="to-year" select="$year"/>
      <xsl:with-param name="to-month" select="$month"/>
      <xsl:with-param name="to-day" select="$day"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:value-of select="format-number($diff + 1, '000')"/>
</xsl:when>

<!-- Номер месяца в виде десятичного числа (01 - 12) -->
<xsl:when test="$code='m'">
  <xsl:value-of select="format-number($month,'00')"/>
</xsl:when>

<!-- Минута в виде десятичного числа (00 - 59) -->
<xsl:when test="$code='M'">
  <xsl:value-of select="format-number($minute,'00')"/>
</xsl:when>

<!-- Индикатор A.M./P.M. в текущей локали для времени в 12-часовом
формате -->
<xsl:when test="$code='p'">
  <xsl:choose>
    <xsl:when test="$hour &lt; 12">AM</xsl:when>
    <xsl:otherwise>PM</xsl:otherwise>
  </xsl:choose>
</xsl:when>

<!-- Секунда в виде десятичного числа (00 - 59) -->
<xsl:when test="$code='S'">
  <xsl:value-of select="format-number($second,'00')"/>
</xsl:when>

<!-- Номер недели в году в виде десятичного числа, когда неделя
начинается с воскресенья (00 - 53) -->
<xsl:when test="$code='U'">
  <!-- add 1 to day -->
  <xsl:call-template name="ckbk:calculate-week-number">
    <xsl:with-param name="year" select="$year"/>
    <xsl:with-param name="month" select="$month"/>
    <xsl:with-param name="day" select="$day + 1"/>
  </xsl:call-template>
</xsl:when>

<!-- День недели в виде десятичного числа (0 - 6; воскресенье - 0) -->
```

```
<xsl:when test="$code='w'">
  <xsl:call-template name="ckbk:calculate-day-of-the-week">
    <xsl:with-param name="year" select="$year"/>
    <xsl:with-param name="month" select="$month"/>
    <xsl:with-param name="day" select="$day"/>
  </xsl:call-template>
</xsl:when>

<!-- Номер недели в году в виде десятичного числа,
когда неделя начинается с понедельника (00 - 53) -->
<xsl:when test="$code='W'">
  <xsl:call-template name="ckbk:calculate-week-number">
    <xsl:with-param name="year" select="$year"/>
    <xsl:with-param name="month" select="$month"/>
    <xsl:with-param name="day" select="$day"/>
  </xsl:call-template>
</xsl:when>

<!-- Представление даты в текущей локали -->
<xsl:when test="$code='x'">
  <xsl:text>[не реализовано]</xsl:text>
</xsl:when>

<!-- Представление времени в текущей локали -->
<xsl:when test="$code='X'">
  <xsl:text>[не реализовано]</xsl:text>
</xsl:when>

<!-- Год без века в виде десятичного числа (00 - 99) -->
<xsl:when test="$code='y'">
  <xsl:value-of select="format-number($year mod 100,'00')"/>
</xsl:when>

<!-- Год с веком в виде десятичного числа -->
<xsl:when test="$code='Y'">
 <xsl:value-of select="format-number($year,'0000')"/>
</xsl:when>

<!-- Название или аббревиатура часового пояса; -->
<!-- если часовой пояс неизвестен, ничего не выводится -->
<xsl:when test="$code='z'">
  <xsl:value-of select="$time-zone"/>
</xsl:when>

<!-- Знак процента -->
```

```
      <xsl:when test="$code='%'">
        <xsl:text>%</xsl:text>
      </xsl:when>

    </xsl:choose>

    <xsl:variable name="remainder"
             select="substring(substring-after($format, '%'), 2)"/>

    <xsl:if test="$remainder">
      <xsl:call-template name="ckbk:format-date-time">
        <xsl:with-param name="year" select="$year"/>
        <xsl:with-param name="month" select="$month"/>
        <xsl:with-param name="day" select="$day"/>
        <xsl:with-param name="hour" select="$hour"/>
        <xsl:with-param name="minute" select="$minute"/>
        <xsl:with-param name="second" select="$second"/>
        <xsl:with-param name="time-zone" select="$time-zone"/>
        <xsl:with-param name="format" select="$remainder"/>
      </xsl:call-template>
    </xsl:if>

</xsl:template>

<xsl:template name="ckbk:format-julian-day">
  <xsl:param name="julian-day"/>
  <xsl:param name="format" select="'%Y-%m-%d'"/>

  <xsl:variable name="a" select="$julian-day + 32044"/>
  <xsl:variable name="b" select="floor((4 * $a + 3) div 146097)"/>
  <xsl:variable name="c" select="$a - floor(($b * 146097) div 4)"/>

  <xsl:variable name="d" select="floor((4 * $c + 3) div 1461)"/>
  <xsl:variable name="e" select="$c - floor((1461 * $d) div 4)"/>
  <xsl:variable name="m" select="floor((5 * $e + 2) div 153)"/>

  <xsl:variable name="day" select="$e - floor((153 * $m + 2) div 5) + 1"/>
  <xsl:variable name="month" select="$m + 3 - 12 * floor($m div 10)"/>
  <xsl:variable name="year" select="$b * 100 + $d - 4800 + floor($m div 10)"/>

  <xsl:call-template name="ckbk:format-date-time">
    <xsl:with-param name="year" select="$year"/>
    <xsl:with-param name="month" select="$month"/>
    <xsl:with-param name="day" select="$day"/>
    <xsl:with-param name="format" select="$format"/>
```

```
    </xsl:call-template>

</xsl:template>
```

XSLT 2.0

Воспользуемся встроенными функциями форматирования даты. В следующем фрагменте предполагается, что дата представлена строкой вида *ГГГГММДД*, что зачастую удобнее на практике, чем формат ISO, которого ожидает функция `format-date`:

```
<xsl:param name="date" as="xs:string"/>

<invoiceDate><xsl:value-of
    select="format-date(
           xs:date(concat(substring($date,1,4),
                          '-',
                          substring($date,5,2),
                          '-',
                          substring($date,7,2))),
                  '[MNn] [D01, [Y0001')"/></invoiceDate>
```

Для даты `$date = '20050811'` мы получим

```
<invoiceDate>August 11, 2005</invoiceDate>
```

Обсуждение

XSLT 1.0

В этом примере мы воспользовались плодами работы, проделанной в предыдущих рецептах. Те виды форматирования, в которых участвует локаль, остались нереализованными, но это можно исправить с помощью функций расширения (см. главу 12).

XSLT 2.0

Чтобы вам было удобнее, приведу выдержку из спецификации W3C.

Предоставляются три функции для представления даты и времени в виде строк с учетом выбранного календаря и локали. У каждой функции есть два варианта:

```
format-dateTime($value as xs:dateTime?,
    $picture as xs:string,
    $date-format-name as xs:string) as xs:string?

format-dateTime($value as xs:dateTime?, $picture as xs:string) as
xs:string?

format-date($value as xs:date?,
    $picture as xs:string,
```

```
    $date-format-name as xs:string) as xs:string?

format-date($value as xs:date?, $picture as xs:string) as xs:string?

format-time($value as xs:time?,
    $picture as xs:string,
    $date-format-name as xs:string) as xs:string?

format-time($value as xs:time?, $picture as xs:string) as xs:string?
```

Функции format-dateTime, format-date и format-time форматируют значение $value в виде строки, используя форматную строку, заданную аргументом $picture, и именованный формат даты, заданный аргументом $date-format-name, или формат даты по умолчанию, если последний аргумент опущен. Результатом функции является форматированное строковое представление аргумента dateTime, date или time.

Все три функции format-dateTime, format-date и format-time собирательно называются *функциями форматирования даты*.

Если имя, заданное аргументом $date-format-name, не является допустимым *QName* или его префикс не был объявлен в объявлении пространства имен, попадающего в область видимости, или таблица стилей не содержит объявления date-format с подходящим expanded-QName, возникает *динамическая ошибка*. Процессор должен либо известить об этой ошибке, либо исправить ее, проигнорировав аргумент $date-format-name. Если процессор способен обнаружить ошибку статически (например, в случае, когда аргумент представляет собой строковый литерал), то он может дополнительно известить о статической ошибке.

Если $value – пустая последовательность, то возвращается пустая последовательность.

Объявление элемента date-format

```
<!-- Категория: объявление -->
<xsl:date-format name = qname
    language = nmtoken
    calendar = qname />
```

Элемент xsl:date-format объявляет *формат даты* (date-format), который предоставляет информацию, используемую функциями форматирования даты. Если атрибут name указан, то элемент объявляет именованный формат даты, в противном случае – формат даты по умолчанию. Значением атрибута name является QName. *Статической ошибкой* считается объявление формата даты по умолчанию или формата с одним и тем же именем более одного раза (даже с разным приоритетом импорта), если только в каждом объявлении значения всех атрибутов не оказываются в точности одинаковыми (принимая во внимание и значения по умолчанию). Если в таблице стилей нет объявления формата даты по умолчанию, то подразумевается объявление, эквивалентное элементу xsl:date-format без каких-либо атрибутов.

Атрибут `language` задает язык для строки, возвращаемой функцией `format-date`. Он должен принимать значение, допустимое для атрибута `xml:lang`. Если этот атрибут опущен, то значение по умолчанию зависит от реализации.

Атрибут `language` используется для выбора следующих зависящих от языка аспектов:

- названий (например, месяцев);
- формы записи чисел;
- соглашения о представлении часа (0-23 или 1-24, 0-11 или 1-12);
- первого дня недели, первой недели месяца.

Набор поддерживаемых языков зависит от реализации.

Атрибут `calendar` определяет календарь, в который должны быть преобразованы значения аргумента `$value` типа `dateTime`, `date` или `time`, и соглашения, применяемые к результирующей строке.

Значение `calendar` должно быть допустимым квалифицированным именем *QName*. Если *QName* не содержит префикса, то подразумевается календарь с одним из описанных ниже обозначений. Если *QName* содержит префикс, то *QName* расширяется в *expanded-QName*, и *expanded-QName* определяет календарь; поведение в этом случае настоящим документом не определено.

Если атрибут `calendar` опущен, используется значение, определяемое локалью.

Статической ошибкой считается ситуация, когда реализация не поддерживает язык, заданный атрибутом `language`, или календарь, заданный атрибутом `calendar`, или то и другое одновременно. Процессор должен либо известить об этой ошибке, либо исправить ее, выбрав вместо указанного зависящее от локали значение того или другого атрибута. Если в итоге будет использован не тот календарь, который был запрошен, то в результирующую строку должно быть включено название календаря.

Перечисленные ниже календари используются последние несколько сотен лет. В прошлом употреблялись и другие календари, которые во многих случаях невозможно поддержать без дополнительных параметров. Такие параметры можно определить с помощью дополнительных атрибутов элемента `xsl:date-format`, квалифицированных префиксом пространства имен. Семантика подобных атрибутов не определяется настоящей спецификацией.

В настоящей спецификации не определяется ни один из перечисленных календарей, а также способы их отображения на пространство значений типа данных xs:date в терминах XML Schema. Существует лишь приближенная эквивалентность между датами, представленными в различных календарях. Например, в разных календарях день начинается не в один и тот же момент, начало дня может также зависеть от географического местоположения. Поэтому реализации, поддерживающие не только григорианский календарь, могут давать различные результаты.

Обозначение	Календарь
AD	Anno Domini (христианская эра)
AH	Anno Hegirae (эра Хиджры)

Обозначение	Календарь
AME	Mauludi Era (количество солнечных лет с момента рождения Мухаммада)
AM	Anno Mundi (еврейская эра от «сотворения мира»)
AP	Anno Persici (персидская эра)
AS	Aji Saka Era (эра Аджи Сака, остров Ява)
BE	Buddhist Era (буддийская эра)
CB	Cooch Behar (эра Кач-Бехар)
CE	Common Era (эра «от рождества Христова»)
CL	Chinese Lunar Era (китайская лунная эра)
CS	Chula Sakarat Era (эра Чула Сакарат, Юго-Восточная Азия)
EE	Ethiopean Era (эфиопская эра)
FE	Fasli Era (эра Фасли)
ISO	календарь ISO 8601
JE	японский календарь
KE	Khalsa Era (сикхский календарь)
KY	Kali Yuga (эра Кали-юга, Индия)
ME	Malabar Era (малабарская эра)
MS	Monarchic Solar Era
NS	Nepal Samwat Era (эра Самвата, Непал)
OS	Old Style («старый стиль», юлианский календарь)
RS	Rattanakosin Era (эра Раттанакосина, Бангкок)
SE	Saka Era (эра Сака, Индия)
SH	мусульманская солнечная эра
SS	эра Сака-Самват
TE	Tripurabda Era (эра Трипури)
VE	Vikrama Era (эра Викрамы)
VS	Vikrama Samvat Era (эра Викрама-Самват)

Должен поддерживаться хотя бы один из перечисленных выше календарей. Какие именно календари поддерживаются, зависит от реализации.

Включенный в список календарь ISO 8601, обозначаемый ISO, по существу не отличается от григорианского календаря AD, но предписывает четко определенные соглашения о нумерации, определенные в стандарте ISO 8601, не подверженные локализации. Точнее, в календаре ISO дни недели нумеруются с 1 (понедельник) по 7 (воскресенье), а первой неделей года считается неделя (с понедельника по воскресенье), содержащая первый четверг года. Числовые значения года, месяца, дня, часа, минуты и секунды в этом календаре такие же, как в лексическом представлении даты и времени, определенном в спецификации XML Schema. Календарь ISO предназначен в первую очередь для приложений, которые порождают дату и время в формате, пригодном для экспорта в другие приложения, а не для человека.

Пространство значений типов данных для даты и времени, определенное в спецификации XML Schema, основано на абсолютных моментах времени. Лексическое пространство для этих типов данных определяет представление абсолютных моментов времени с помощью пролептического григорианского календаря, то есть современного западного календаря, экстраполированного

в прошлое и будущее, однако пространство значений от календаря не зависит. Функции форматирования даты порождают представление одной и той же абсолютной временной точки в указанном календаре. Так, например, дата, лексическое представление которой в соответствии с XML Schema равно 1502-01-11 (день рождения папы Григория XIII), в календаре Old Style (юлианском) может быть представлена как 1 January 1502. Это отражает тот факт, что между григорианским и юлианским календарями есть разница в десять дней. Было бы неправильно (и привело бы к неправильным результатам) представлять эту дату в элементе или атрибуте типа xs:date как 1502-01-01, пусть даже именно так она записывается в современных документах.

Форматная строка

Форматная строка состоит из последовательности специальных маркеров и литеральных подстрок. Подстрока, заключенная в фигурные скобки, интерпретируется как маркер; подстроки, находящиеся вне квадратных скобок, считаются литеральными. Литеральные подстроки могут отсутствовать, а если имеются, то выводятся без изменения, в том числе со всеми пропусками. Если внутри литеральной подстроки встречается открывающая или закрывающая квадратная скобка, она должна быть повторена дважды. Вместо маркеров в результирующую строку подставляются строки, описывающие те или иные аспекты форматируемой даты или времени. Маркеры подробно описаны ниже.

Маркер состоит из *спецификатора компонента*, за которым может следовать один или два модификатора представления и/или необязательный модификатор длины. Любые пробелы внутри маркера игнорируются.

Спецификатор компонента обозначает нужный компонент даты или времени и может принимать следующие значения:

Спецификатор	Назначение	Модификатор представления по умолчанию
Y	Год	1
M	Месяц года	1
D	День месяца	1
d	День года	1
F	День недели	n
W	Неделя года	1
w	Неделя месяца	1
H	Час суток (24 часа)	1
h	Час полусуток (12 часов)	1
P	Маркер a.m./p.m.	n
m	Минута часа	1
s	Секунда минуты	1
f	Дробная доля секунды	1

Спецификатор	Назначение	Модификатор представления по умолчанию
Z	Часовой пояс в виде смещения от UTC или, если задан алфавитный модификатор, стандартное название часового пояса (например, PST)	1
z	Часовой пояс в виде смещения от GMT; например, GMT+1	1
C	Календарь – название или аббревиатура	n
E	Эра – название точки отсчета при нумерации годов, например, начало правления монарха	n

Динамической ошибкой считается ситуация, когда спецификатор компонента в форматной строке ссылается на компоненты, отсутствующие в переданном параметре `$value` или не поддерживаемые выбранным календарем. Это *исправимая ошибка*. Процессор может известить об ошибке или исправить ее, проигнорировав некорректные спецификаторы компонентов.

Первый *модификатор представления* определяет стиль представления значения компонента и может принимать следующие значения:

Модификатор	Назначение
A	Буквы, верхний регистр
a	Буквы, нижний регистр (может начинаться с заглавной буквы, если того требует язык)
N	Название, верхний регистр
n	Название, нижний регистр (может начинаться с заглавной буквы, если того требует язык)
цифра 1	Десятичное представление
i	Римские числительные в нижнем регистре
I	Римские числительные в верхнем регистре

Любой символ с десятичным значением 1 (в смысле кодировки Unicode) генерирует десятичное представление числа с помощью подходящего набора цифр Unicode.

Любой другой символ можно использовать для указания последовательности нумерации, начинающейся в этого символа, если реализация поддерживает такую последовательность нумерации.

Если реализация не поддерживает указанный модификатор представления, то вместо него должен использоваться модификатор представления, подразумеваемый по умолчанию для данного компонента.

Если первый модификатор представления присутствует, то за ним может следовать второй модификатор представления, который может принимать следующие значения:

Модификатор	Назначение
t	Традиционная нумерация. То же самое, что `letter-value="traditional"` в элементе `xsl:number`.
o	Порядковая форма числа, например: 3rd или 8°.

Вне зависимости от того, включен ли модификатор представления, можно указать модификатор ширины. Он говорит о том, сколько символов или цифр должно быть в представлении значения.

Модификатор ширины имеет вид:

```
min-width ("-" max-width)?
```

где `min-width` – либо целое без знака, обозначающее минимальное число выводимых символов, либо символ `*`, говорящий о том, что минимум явно не задан, а `max-width` – либо целое без знака, обозначающее максимальное число выводимых символов, либо символ `*`, говорящий о том, что максимум явно не задан. Если часть `max-width` опущена, подразумевается `*`. Оба целых числа, если они присутствуют, должны быть положительны.

Если модификатор ширины не задан, то выводится столько символов, сколько необходимо для представления значения компонента без обрезания и дополнения; результат называется *полным представлением* значения.

Если число символов в полном представлении превышает заданную максимальную ширину, то рекомендуется, чтобы процессор подыскал альтернативное более короткое представление, которое уместилось бы в заданную максимальную ширину. Если модификатор представления равен `n` или `N`, то это достигается сокращением имени либо путем использования общепринятой аббревиатуры, либо путем отбрасывания символов справа. Так, если `max-width` равно 4, то следует брать четырехбуквенные аббревиатуры, но использование трехбуквенных тоже допустимо, если таково стандартное соглашение. (Например, «Tuesday» можно сократить до «Tues», а «Friday» – до «Fri».) В случае компонента «год» задание `max-width` означает, что нужно опустить старшие цифры года; например, если `max-width` равно 2, то год 2003 должен быть выведен как 03. Если не существует способа уместить значение в заданную максимальную ширину (например, при использовании римских цифр), должно выводиться полное представление значения.

Если полное представление значения короче заданной минимальной ширины, то рекомендуется, чтобы процессор дополнил значение до заданной ширины. В случае десятичного представления числа это следует делать путем дописывания в начало нулей из подходящего набора символов. В остальных случаях в начало дописываются пробелы.

Выбор названий и сокращений для заданного языка *зависит от реализации*. Например, в одной реализации название месяца July (июль) может сокращаться до Jul, а в другой – до Jly. В случае немецкого языка суббота может называться как Samstag, так и Sonnabend. Реализация может предоставлять механизмы, позволяющие пользователю управлять выбором названия или аббревиатуры в таких случаях.

В примерах ниже показаны варианты форматирования даты и времени. Предполагается, что используется григорианский календарь и что значение атрибута `name` в объявлении `xsl:date-format` в таблице стилей такое же, как значение атрибута `language`. (Например, `<date-format name="sv" language="sv"/>`.)

Результирующее значение	Выражение
2002-12-31	format-date($d,"[Y]-[M]-[D]")
12-31-2002	format-date($d,"[M]-[D]-[Y]")
31-12-2002	format-date($d,"[D]-[M]-[Y]")
31 XII 2002	format-date($d,"[D1] [MI] [Y]")
31st December, 2002	format-date($d,"[Do] [Mn], [Y]", "en")
31 DEC 2002	format-date($d,"[D] [MN,*-3] [Y]", "en")
December 31, 2002	format-date($d,"[Mn] [D], [Y]", "en")
31 Dezember, 2002	format-date($d,"[D] [Mn], [Y]", "de")
	Tisdag 31 December 2002
	format-date($d,"[Fn] [D] [Mn] [Y]", sv")
[2003-04-07]	format-date($d,"[[[Y]-[M]-[D]]]")
3:58 PM	format-time($t,"[h]:[m] [PN]", "en")
3:58:45 pm	format-time($t,"[h]:[m]:[s] [Pn]", "en")
3:58:45 PM PDT	format-time($t,"[h]:[m]:[s] [PN] [ZN,*-3]", "en")
3:58:45 o'clock PM PDT	format-time($t,"[h]:[m]:[s] o'clock [PN] [ZN,*-3]", "en")
15:58	format-time($t,"[H]:[m]")
15:58:45	format-time($t,"[H]:[m]:[s]")
15:58:45 GMT+02:00	format-time($t,"[H]:[m]:[s] [z]". "en")
15.58 Uhr GMT+02:00	format-time($t,"[H].[m] Uhr [z]". "de")

В следующих примерах используются не-григорианские календари.

```
<!-- Пример: тайский календарь -->
<xsl:date-format name="modern_Thai" language="th" calendar="BE"/>
format-date($d, "[D&#x0E51;] [Mn] [Y&#x0E51;]", "modern_Thai")
<!-- Результат:
๓๑ ธันวาคม ๒๕๔๕ -->

<!-- Пример: исламский календарь -->
<xsl:date-format name="Islamic" language="ar" calendar="AH"/>
format-date($d, "[D&#x0661;] [Mn] [Y&#x0661;]", "Islamic")

<!-- Результат: ٢٦ شوّال ١٤٢٣ -->

<!-- Пример: еврейский календарь -->
<xsl:date-format name="Jewish" language="he" calendar="AM"/>
format-date($d, "[D] [Mn] [Y]", "Jewish")
<!-- Результат: 26 טבת 5763 -->
```

4.11. Определение светских и церковных праздников

Задача

Требуется узнать, является ли заданная дата праздничной.

Решение

К первому типу праздников относятся те, которые каждый год отмечаются в один тот же день. Например, определить абсолютную дату Дня независимости в США совсем просто:

```
<xsl:template name="ckbk:independence-day">
    <xsl:param name="year"/>
    <xsl:call-template name="ckbk:date-to-absolute-day">
           <xsl:with-param name="month" select="7"/>
           <xsl:with-param name="day" select="4"/>
           <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
</xsl:template>
```

Ко второму типу относятся праздники, отмечаемые в один и тот же день недели, отсчитываемой от начала или конца месяца. Такие даты можно проверить с помощью следующей утилиты, которая обертывает шаблон `k-day-on-or-before-abs-day` из рецепта 4.8:

```
<xsl:template name="ckbk:n-th-k-day">
    <!-- n-ое вхождение k в данном месяце -->
    <!-- Если n положительно, отсчет ведется от начала месяца,
    иначе — от конца-->
    <xsl:param name="n"/>
    <!-- k - день недели (0 = воскресенье) -->
    <xsl:param name="k"/>

    <xsl:param name="month"/>
    <xsl:param name="year"/>

    <xsl:choose>
       <xsl:when test="$n > 0">
           <xsl:variable name="k-day-on-or-before">
           <xsl:variable name="abs-day">
              <xsl:call-template name="ckbk:date-to-absolute-day">
                   <xsl:with-param name="month" select="$month"/>
                   <xsl:with-param name="day" select="7"/>
                   <xsl:with-param name="year" select="$year"/>
```

```
            </xsl:call-template>
          </xsl:variable>
          <xsl:call-template name="ckbk:k-day-on-or-before-abs-day">
             <xsl:with-param name="abs-day" select="$abs-day"/>
             <xsl:with-param name="k" select="$k"/>
          </xsl:call-template>
        </xsl:variable>
      <xsl:value-of select="$k-day-on-or-before + 7 * ($n - 1)"/>
   </xsl:when>
   <xsl:otherwise>
      <xsl:variable name="k-day-on-or-before">
         <xsl:variable name="abs-day">
          <xsl:call-template name="ckbk:date-to-absolute-day">
              <xsl:with-param name="month" select="$month"/>
              <xsl:with-param name="day">
                 <xsl:call-template name="ckbk:last-day-of-month">
                  <xsl:with-param name="month" select="$month"/>
                  <xsl:with-param name="year" select="$year"/>
                  </xsl:call-template>
               </xsl:with-param>
               <xsl:with-param name="year" select="$year"/>
            </xsl:call-template>
          </xsl:variable>
          <xsl:call-template name="ckbk:k-day-on-or-before-abs-day"
             <xsl:with-param name="abs-day" select="$abs-day"/>
             <xsl:with-param name="k" select="$k"/>
          </xsl:call-template>
        </xsl:variable>
        <xsl:value-of select="$k-day-on-or-before + 7 * ($n + 1)"/>
      </xsl:otherwise>
   </xsl:choose>
  </xsl:template>
```

Здесь предполагается использование григорианского календаря. Если вам нужно вычислять относительные даты в других календарях, придется написать соответствующий шаблон самостоятельно.

Теперь легко решить задачу для таких праздников, как День труда и День поминовения в США (соответственно первый понедельник сентября и последний понедельник мая):

```
<xsl:template name="ckbk:labor-day">
    <xsl:param name="year"/>
    <xsl:call-template name="ckbk:n-th-k-day">
       <xsl:with-param name="n" select="1"/>
       <xsl:with-param name="k" select="1"/>
```

```
        <xsl:with-param name="month" select="9"/>
        <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="ckbk:memorial-day">
    <xsl:param name="year"/>
    <xsl:call-template name="ckbk:n-th-k-day">
        <xsl:with-param name="n" select="-1"/>
        <xsl:with-param name="k" select="1"/>
        <xsl:with-param name="month" select="5"/>
        <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
</xsl:template>
```

Хотя дни перехода на летнее и зимнее время в США – не праздники, их тоже легко вычислить:

```
<xsl:template name="ckbk:day-light-savings-start">
    <xsl:param name="year"/>
    <xsl:call-template name="ckbk:n-th-k-day">
        <xsl:with-param name="n" select="1"/>
        <xsl:with-param name="k" select="0"/>
        <xsl:with-param name="month" select="4"/>
        <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="ckbk:day-light-savings-end">
    <xsl:param name="year"/>
    <xsl:call-template name="ckbk:n-th-k-day">
        <xsl:with-param name="n" select="-1"/>
        <xsl:with-param name="k" select="0"/>
        <xsl:with-param name="month" select="10"/>
        <xsl:with-param name="year" select="$year"/>
    </xsl:call-template>
</xsl:template>
```

Обсуждение

Рассмотреть все светские и церковные праздники в каждой стране и в каждом году невозможно. Однако их можно разбить на две категории: отмечаемые каждый год в один и тот же день (например, День независимости в США) и отмечаемые в некоторый день недели, отсчитываемой от начала или конца определенного месяца (например, День труда в США). Даты церковных праздников часто легко

определяются в соответствующем им календаре, но с трудом вычисляются в других календарных системах. Например, в восточной православной церкви Рождество отмечается 25 декабря по юлианскому календарю. Следовательно, по григорианскому календарю православное рождество иногда празднуется в начале года, иногда в конце, а иногда не празднуется вовсе. Поскольку мы не в состоянии обсудить все церковные праздники для всех религий, заинтересованному читателю предлагается обратиться к справочным материалам, упомянутым в начале этой главы.

Глава 5. Отбор и обход

Я беру глыбу мрамора и отсекаю все лишнее.
Франсуа-Огюст Роден

Я обошел эту страну вдоль и поперек,
говорил с лучшими людьми и могу
вас уверить, что обработка данных –
преходящее увлечение, которое
не продлится и года.
Главный редактор отдела
деловой литературы крупного
издательства, 1957 год

5.0. Введение

Любая сколько-нибудь интересная задача в XSLT подразумевает выполнение двух взаимосвязанных операций: определение того, какие узлы XML-документа следует посетить (*отбор*), и в каком порядке (*обход*). Задача отбора решается в основном средствами языка XPath, который описывается отдельной спецификацией, тесно связанной с XSLT. Обход реализуется встроенными в XSLT конструкциями и зависит от того, как вы организуете шаблоны, в которых эти конструкции используются.

Знатоки XSLT вряд ли найдут что-то новое для себя в этой главе. И все же она важна по двум причинам. Во-первых, в ней изложены идеи, которые отличают XSLT от других языков программирования. Именно на них спотыкаются новички, делающие первые попытки овладеть XSLT. Во-вторых, представленные примеры – это кирпичики, из которых строятся более сложные рецепты в последующих главах. Практически все, что делается на языке XSLT, включает в том или ином виде отбор и обход. Если прибегнуть к аналогии с кулинарией, то умение приготовить хороший коричневый основной соус – необходимое условие приготовления соуса эспаньоль![1] Эта глава содержит рецепты различных вариантов «основного соуса», которыми вы должны овладеть в совершенстве, прежде чем переходить к более сложным приложениям XSLT.

[1] Соус эспаньоль – это концентрированный коричневый основной соус, смешанный с томатами и поджаренной в масле мукой, а затем сильно уваренный. Основной соус готовится из телячьей и говяжьей голяшки. Соус эспаньоль необходим для приготовления деми-глясс – исходного материала для многих других коричневых соусов. Так что в деле «повторного использования» повара легко побивают разработчиков программ!

Хотя примеры в этой главе довольно просты, ее нельзя считать руководством по XPath или XSLT. Читатель должен быть знаком с основами XPath, изложенными в главе 1. Предполагается также, что вы знаете о правилах обработки по умолчанию, применяемых в XSLT, и о том, как XSLT решает, какой шаблон обрабатывать далее. Если вам нужна дополнительная информация по этим темам, рекомендую книгу Майкла Кэя *XSLT Programmer's Reference* (Wrox Press, 2004) или Эвана Ленца *XSLT 1.0 Pocket Reference* (O'Reilly, 2005). В этой главе различия между XSLT 1.0 и 2.0 не будут обсуждаться так подробно, как в предыдущих. Объясняется это тем, что в версии XSLT 1.0 средства выражения описываемых операций и так достаточно развиты. Исключение составляют случаи, когда отбор основывается на построении групп узлов, связанных между собой условиями, не выражаемыми в терминах иерархической организации XML-документа. Так, желанным дополнением стала новая команда `xsl:for-each-group`.

В некоторых примерах из этой главы применяется терминология, характерная для алгоритмов работы с древовидными структурами. Новичкам, возможно, не вполне ясно, какое отношение деревья имеют к XML. Ответ прост: XML можно рассматривать как язык описания деревьев, а XSLT – как язык обработки описанных таким образом деревьев. Вообще-то, процессор XSLT может обрабатывать также входные данные, представленные не в виде XML-документа; надо лишь «убедить» процессор в том, что это дерево, воспользовавшись SAX-драйвером. Майкл Кэй демонстрирует такую технику в книге *XSLT Programmer's Reference* (Wrox Press, 2001). Похожие примеры приводит Эрик М. Бурке (Eric M. Burke) в книге *Java and XSLT* (O'Reilly, 2001). Так или иначе, за примерами из этой главы стоит важная идея о том, что взгляд на задачу с точки зрения деревьев открывает несколько полезных приемов обработки XML.

В примерах ниже мы опираемся на следующие два тестовых документа. Чтобы не повторяться в каждом рецепте, я привожу их в примерах 5.1 и 5.2.

Пример 5.1. SalesByPerson.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<salesBySalesperson>
  <salesperson name="John Adams" seniority="1">
     <product sku="10000" totalSales="10000.00"/>
     <product sku="20000" totalSales="50000.00"/>
     <product sku="25000" totalSales="920000.00"/>
  </salesperson>
  <salesperson name="Wendy Long" seniority="5">
     <product sku="10000" totalSales="990000.00"/>
     <product sku="20000" totalSales="150000.00"/>
     <product sku="30000" totalSales="5500.00"/>
  </salesperson>
  <salesperson name="Willie B. Aggressive" seniority="10">
     <product sku="10000" totalSales="1110000.00"/>
     <product sku="20000" totalSales="150000.00"/>
     <product sku="25000" totalSales="2920000.00"/>
```

```
      <product sku="30000" totalSales="115500.00"/>
      <product sku="70000" totalSales="10000.00"/>
  </salesperson>
  <salesperson name="Arty Outtolunch" seniority="10"/>
</salesBySalesperson>
```

Пример 5.2. orgchart.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<employee name="Jil Michel" sex="female">
    <employee name="Nancy Pratt" sex="female">
       <employee name="Phill McKraken" sex="male"/>
       <employee name="Ima Little" sex="female">
          <employee name="Betsy Ross" sex="female"/>
       </employee>
    </employee>
    <employee name="Jane Doe" sex="female">
       <employee name="Walter H. Potter" sex="male"/>
       <employee name="Wendy B.K. McDonald" sex="female">
            <employee name="Craig F. Frye" sex="male"/>
            <employee name="Hardy Hamburg" sex="male"/>
            <employee name="Rich Shaker" sex="male"/>
       </employee>
    </employee>
    <employee name="Mike Rosenbaum" sex="male">
       <employee name="Cindy Post-Kellog" sex="female">
            <employee name="Allen Bran" sex="male"/>
            <employee name="Frank N. Berry" sex="male"/>
            <employee name="Jack Apple" sex="male"/>
       </employee>
       <employee name="Oscar A. Winner" nsex="male">
            <employee name="Jack Nickolas" sex="male">
            <employee name="R.P. McMurphy" sex="male"/>
       </employee>
            <employee name="Tom Hanks" sex="male">
               <employee name="Forest Gump" sex="male"/>
               <employee name="Andrew Beckett" sex="male"/>
            </employee>
            <employee name="Susan Sarandon" sex="female">
               <employee name="Helen Prejean" sex="female"/>
            </employee>
       </employee>
    </employee>
</employee>
```

5.1. Игнорирование элементов-дубликатов

Задача

Требуется отобрать все узлы, уникальные в данном контексте, используя заданный критерий уникальности.

Решение

XSLT 1.0

Задача отбора уникальных узлов – типичное применение осей `preceding` и `preceding-sibling`. Если не все отбираемые элементы являются братьями, пользуйтесь осью `preceding`. Следующий код порождает список уникальных продуктов, встречающихся в документе *SalesByPerson.xml*:

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

<xsl:template match="/">

<products>
  <xsl:for-each select="//product[not(@sku=preceding::product/@sku)]">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</products>
</xsl:template>

</xsl:stylesheet>
```

Если все элементы являются братьями, пользуйтесь осью `preceding-sibling`:

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
   <product sku="10000" totalSales="10000.00"/>
   <product sku="10000" totalSales="990000.00"/>
   <product sku="10000" totalSales="1110000.00"/>
   <product sku="20000" totalSales="50000.00"/>
   <product sku="20000" totalSales="150000.00"/>
   <product sku="20000" totalSales="150000.00"/>
   <product sku="25000" totalSales="920000.00"/>
   <product sku="25000" totalSales="2920000.00"/>
   <product sku="30000" totalSales="5500.00"/>
   <product sku="30000" totalSales="115500.00"/>
   <product sku="70000" totalSales="10000.00"/>
</products>
```

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

<xsl:template match="/products">
<products>
  <xsl:for-each select="product[not(@sku=preceding-sibling::product/@sku)]">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</products>
</xsl:template>

</xsl:stylesheet>
```

Чтобы избежать не всегда эффективного использования оси `preceding`, поднимитесь вверх до предков, которые *являются* братьями, а затем воспользуйтесь осью `preceding-sibling` и спуститесь до узлов, подлежащих проверке:

```
<xsl:for-each select="//product[not(@sku=../preceding-sibling::*/
                        product/@sku)]">
  <xsl:copy-of select="."/>
</xsl:for-each>
```

Если вы точно знаете, что элементы отсортированы и, значит, узлы-дубликаты расположены по соседству (как в приведенном выше документе `products`), то можете ограничиться рассмотрением лишь непосредственно предшествующего брата:

```
<xsl:for-each
     select="/salesperson/product[not(@name=preceding-
                                      sibling::product[1]/@name)]">
    <!-- сделать что-то с каждым продуктом с уникальным названием -->
</xsl:for-each>
```

XSLT 2.0

В XSLT 2.0 эту задачу можно рассматривать как задачу группировки. Для решения достаточно применить команду `for-each-group` и задать критерий уникальности с помощью атрибута `group-by`. В качестве уникального берем первый узел в каждой группе:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <products>
      <xsl:for-each-group select="//product" group-by="@sku">
         <xsl:copy-of select="current-group()[1]"/>
      </xsl:for-each-group>
```

```
    </products>
  </xsl:template>

</xsl:stylesheet>
```

Обсуждение

XSLT 1.0

С помощью функции расширения `node-set()` можно также решить задачу следующим образом:

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

   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="/">

   <xsl:variable name="products">
     <xsl:for-each select="//product">
         <xsl:sort select="@sku"/>
         <xsl:copy-of select="."/>
     </xsl:for-each>
   </xsl:variable>

   <products>
     <xsl:for-each select="exsl:node-set($products)/product">
         <xsl:variable name="pos" select="position()"/>
         <xsl:if test="$pos = 1 or
          not(@sku = $products/preceding-sibling::product[1]/@sku">
           <xsl:copy-of select="."/>
         </xsl:if>
     </xsl:for-each>
   </products>
</xsl:template>

</xsl:stylesheet>
```

Я не встречал реализации, в которой эта техника оказалась бы быстрее, чем использование оси `preceding`. Однако у нее есть преимущества в случаях, когда проверка на дубликат не тривиальна. Рассмотрим, например, ситуацию, где для определения одинаковости необходимо конкатенировать два атрибута:

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

```
  xmlns:exslt="http://exslt.org">

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">

<xsl:variable name="people">
    <xsl:for-each select="//person">
       <xsl:sort select="concat(@lastname,@firstname)"/>
       <xsl:copy-of select="."/>
    </xsl:for-each>
</xsl:variable>

<products>
    <xsl:for-each select="exsl:node-set($people)/person">
       <xsl:variable name="pos" select="position()"/>
       <xsl:if test="$pos = 1 or
              concat(@lastname,@firstname) !=
                          concat(people/person[$pos - 1]/@lastname,
                                 people/person[$pos - 1]/@firstname)">
              <xsl:copy-of select="."/>
           </xsl:if>
    </xsl:for-each>
</products>

</xsl:template>
```

Следующая попытка удалить дубликаты работать *не* будет:

```
<xsl:template match="/">
<products>
    <xsl:for-each select="//product[not(@sku=preceding::product[1]/@sku)]">
           <xsl:sort select="@sku"/>
           <xsl:copy-of select="."/>
    </xsl:for-each>
</products>
</xsl:template>
```

Не пытайтесь сортировать для того, чтобы избежать рассмотрения всех элементов, кроме непосредственного предшественника. По оси `preceding` узлы расположены в исходном порядке документа. То же касается и оси `preceding-sibling`. Следующий код также заведомо неработоспособен:

```
<xsl:template match="/">

<xsl:variable name="products">
    <xsl:for-each select="//product">
```

```
    <!-- сортировка удалена -->
        <xsl:copy-of select="."/>
    </xsl:for-each>
</xsl:variable>

<products>
    <xsl:for-each select="exsl:node-set($products)/product">
       <xsl:sort select="@sku"/>
       <xsl:variable name="pos" select="position()"/>
       <xsl:if test="$pos = 1 or
             @sku != $products/product[$pos - 1]/@sku">
          <xsl:copy-of select="."/>
       </xsl:if>
    </xsl:for-each>
</products>
</xsl:template>
```

Этот код не работает, так как `position()` возвращает позицию после сортировки, но отсортировано было не содержимое `$products`, а содержимое недоступной копии.

XSLT 2.0

Иногда нужно удалять только соседние дубликаты. Рассмотрим, к примеру, набор данных, полученный в результате последовательности измерений в определенные моменты времени. Если измеряемая система ведет себя стабильно, то результаты многих соседних измерений будут одинаковы. Было бы хорошо избавиться от соседних дубликатов, не удаляя такие же результаты, если они появляются в последовательности позже.

Для решения этой задачи тоже подойдет команда `xsl:for-each-group`, только с атрибутом `group-adjacent`, а не `group-by`:

```
<measurements>
  <data time="12:00:00" value="1.0"/>
  <data time="12:00:01" value="1.0"/>
  <data time="12:00:02" value="1.1"/>
  <data time="12:00:03" value="1.1"/>
  <data time="12:00:04" value="1.0"/>
  <data time="12:00:05" value="1.1"/>
  <data time="12:00:06" value="1.2"/>
  <data time="12:00:07" value="1.3"/>
  <data time="12:00:08" value="1.4"/>
  <data time="12:00:09" value="1.6"/>
  <data time="12:00:10" value="1.9"/>
  <data time="12:00:11" value="2.1"/>
  <data time="12:00:12" value="1.7"/>
  <data time="12:00:13" value="1.5"/>
```

```
</measurements>

<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/measurements">
    <xsl:copy>
      <xsl:for-each-group select="data" group-adjacent="@value">
        <xsl:copy-of select="current-group()[1]"/>
      </xsl:for-each-group>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>

<!-- Результат -->
<measurements>
  <data time="12:00:00" value="1.0"/>
  <data time="12:00:02" value="1.1"/>
  <data time="12:00:04" value="1.0"/>
  <data time="12:00:05" value="1.1"/>
  <data time="12:00:06" value="1.2"/>
  <data time="12:00:07" value="1.3"/>
  <data time="12:00:08" value="1.4"/>
  <data time="12:00:09" value="1.6"/>
  <data time="12:00:10" value="1.9"/>
  <data time="12:00:11" value="2.1"/>
  <data time="12:00:12" value="1.7"/>
  <data time="12:00:13" value="1.5"/>
</measurements>
```

См. также

В FAQe по XSLT (*http://www.dpawson.co.uk/xsl/sect2/N2696.html*) описывается решение этой задачи с использованием ключей, а также приводятся решения родственных задач.

5.2. Отбор всех элементов, кроме одного

Задача

Требуется отобрать все элементы в определенном контексте за исключением некоторых.

Решение

Самый лучший способ отобрать все элементы, кроме одного, такой:

```
<xsl:apply-templates select="*[not(self::element-to-ignore)]"/>
```

или то же самое, но посредством итерации:

```
<xsl:for-each select="*[not(self::element-to-ignore)]">
...
</xsl:for-each>
```

Обсуждение

Когда неопытный программист на XSLT в первый раз пытается отобрать все элементы, кроме одного, ему на ум чаще всего приходит такая конструкция:

```
<xsl:apply-templates select="*[name() != 'element-to-ignore']"/>
```

Во многих случаях такой код будет работать, проблемы возникнут, когда в документе используются пространства имен. Напомню, что функция `name()` возвращает квалифицированное имя узла (QName), то есть префикс пространства имен, конкатенированный с локальной частью имени. Однако никто не заставляет автора XML-документа использовать конкретный префикс.

```
<!-- Не будет работать, если автор написал SALES:product вместо
sales:product -->
<xsl:apply-templates select="*[name() != 'sales:product']"/>
```

Можно было бы воспользоваться функцией `local-name()`. Но тогда были бы проигнорированы элементы из всех пространств имен с заданным локальным именем. Вряд ли это то, что вы хотите получить.

Эта рекомендация относится только к элементам, а не к атрибутам. Если нужно отобрать все атрибуты, кроме одного, смело используйте `local-name()`. Если речь идет об именах, то ось `self` содержит только элементы. Иными словами, пишите `<xsl:copy-of select=@*[local-name() != 'ignored-attribute']/>`, а не `<xsl:copy-of select=@*[not(self::ignored-attribute)]/>`.

Наконец, чтобы избежать путаницы, отметим, что отбор всех элементов, кроме одного, – не то же самое, что отбор всех элементов, кроме одного *экземпляра* элемента. Вторая задача решается так, как показано в одном из ранее встречавшихся примеров:

```
<xsl:apply-templates select="*[generate-id() != generate-id($node-to-ignore)]"/>
```

См. также

В книге Дженни Теннисон *XSLT and XPath on the Edge* (M&T Books, 2001) подробно рассматривается вопрос о том, когда следует и не следует применять функции `name()` и `local-name()`.

5.3. Отбор узлов по контексту

Задача

Требуется отобрать узлы, лежащие между двумя заданными.

Решение

XSLT 1.0

В XSLT 1.0 есть несколько способов решить эту задачу. Самый простой для понимания заключается в вычислении позиций узлов, которые должны отбираться на каждом шаге. Предположим, что имеется такой документ:

```
<doc>
    <heading>Да не нужна мне никакая структура</heading>
    <para>1.1</para>
    <para>1.2</para>
    <para>1.3</para>
    <para>1.4</para>
    <heading>Признания обитателя плоской земли</heading>
    <para>2.1</para>
    <para>2.2</para>
    <para>2.3</para>
    <heading>Плоские иерархии сохраняют леса!</heading>
    <para>3.1</para>
    <para>3.2</para>
</doc>
<xsl:template match="/doc">
    <xsl:copy>
       <!-- Сначала отбираем ограничивающие элементы -->
       <xsl:apply-templates select="heading"/>
    </xsl:copy>
</xsl:template>

<!-- Сопоставление с ограничивающими элементами -->
  <xsl:template match="heading">
  <!-- Вычисляем, сколько интересующих нас элементов (para) следует за
  этим заголовком -->
  <xsl:variable name="numFollowingPara"
       select="count(following-sibling::para)"/>

    <!-- Вычисляем, сколько интересующих нас элементов (para) следует за
    следующим заголовком, и вычитаем это значение из полученного ранее
    счетчика, чтобы узнать позицию последнего абзаца в данной группе -->
    <xsl:variable name="lastParaInHeading"
       select="$numFollowingPara -
            count(following-sibling::heading[1]/following-sibling::para)"/>

    <!-- Теперь можно отобрать нужные элементы по позиции относительно
    текущего заголовка -->

    <xsl:apply-templates
```

```
        select="following-sibling::para[position() &lt;= $lastParaInHeading]"/>

    </xsl:template>
```

XSLT 2.0

Эта задача как будто специально придумана для демонстрации команды `for-each-group`. В данном случае мы воспользуемся атрибутом `group-starting-with`:

```
<xsl:template match="/doc">
  <xsl:copy>
    <xsl:for-each-group select="*" group-starting-with="heading">
      <!-- Отобрать элементы para в группе, ограниченной с двух сторон
       элементами heading -->
       <xsl:apply-templates select="current-group()[self::para]"/>
    </xsl:for-each-group>
  </xsl:copy>
</xsl:template>
```

Обсуждение

Отбор узлов на основе их позиции относительно других узлов – типичная задача, возникающая, когда нужно преобразовывать XML-документы, где структура подразумевается неявно, а не закодирована прямо в иерархии. Ясно, что если бы каждая группа, состоящая из заголовка и относящихся к нему абзацев, содержалась внутри отдельного родительского элемента (например, `section`), то задача была бы тривиальна. Здесь мы имеем классический компромисс между простотой документа с точки зрения его создателя и с точки зрения того, кто должен его преобразовывать. С появлением в XSLT 2.0 команды `for-each-group` обрабатывать слабо структурированные документы стало гораздо проще.

См. также

В рецепте 8.8 показаны применения описанной техники для преобразования неявно структурированных документов в явно структурированные. Там же приведены другие подходы к решению этой задачи в XSLT 1.0 и 2.0.

5.4. Выполнение обхода в прямом порядке

Задача

Требуется рекурсивно обработать сначала сам элемент, а потом его потомков.

Решение

Все решения в этом рецепте имеют такой общий вид:

```
<xsl:template match="node()">
    <!-- Сделать что-то с текущим узлом -->

    <!-- Обработать потомков -->
    <xsl:apply-templates/>
</xsl:template>
```

Обсуждение

Термин *прямой порядок* применяется, когда речь идет об обходе дерева таким образом, что сначала посещается корневой узел, а затем рекурсивно его дочерние узлы. Пожалуй, это самый распространенный способ обработки XML-документов. Из многочисленных применений этой идиомы мы в настоящей главе рассмотрим только два. В последующих рецептах мы будем часто сталкиваться с таким способом обхода.

Рассмотрим упрощенную структурную схему организации (пример 5.2), в которой B является дочерним элементом A, если работник, представленный элементом B, непосредственно подчиняется работнику, представленному элементом A. В примере 5.3 приведена программа, которая с помощью обхода в прямом порядке поясняет, кто кем руководит. В примере 5.4 показано, что печатается в результате работы этой программы.

Пример 5.3. Таблица стилей

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

<xsl:output method="text"/>
<xsl:strip-space elements="*"/>

<xsl:template match="/employee" priority="10">
   <xsl:value-of select="@name"/><xsl:text> глава компании. </xsl:text>
   <xsl:call-template name="HeShe"/><xsl:text> руководит </xsl:text>
   <xsl:call-template name="manages"/>
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="employee[employee]">
   <xsl:value-of select="@name"/><xsl:text> менеджер. </xsl:text>
   <xsl:call-template name="HeShe"/> <xsl:text> руководит </xsl:text>
   <xsl:call-template name="manages"/>
   <xsl:apply-templates/>
</xsl:template>

<xsl:template match="employee">
    <xsl:value-of select="@name"/><xsl:text> не имеет подчиненных. </xsl:text>
   <xsl:text>&#xa;</xsl:text>
```

```
</xsl:template>

<xsl:template name="HeShe">
   <xsl:choose>
    <xsl:when test="@sex = 'male' ">
       <xsl:text>Он</xsl:text>
    </xsl:when>
    <xsl:otherwise>
       <xsl:text>Она</xsl:text>
    </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<xsl:template name="manages">
   <xsl:for-each select="*">
    <xsl:choose>
       <xsl:when test="position() != last() and last() > 2">
           <xsl:value-of select="@name"/><xsl:text>, </xsl:text>
       </xsl:when>
       <xsl:when test="position() = last() and last() > 1">
            <xsl:text> and </xsl:text:value-of
                      select="@name"/><xsl:text> и </xsl:text>
       </xsl:when>
       <xsl:when test="last() = 1">
           <xsl:value-of select="@name"/><xsl:text>. </xsl:text>
       </xsl:when>
       <xsl:otherwise>
           <xsl:value-of select="@name"/>
       </xsl:otherwise>
       </xsl:choose>
    </xsl:for-each>
   <xsl:text>&#xa;&#xa;</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Пример 5.4. Результат

```
Jil Michel глава компании. Она руководит Nancy Pratt, Jane Doe, и Mike
Rosenbaum.

Nancy Pratt менеджер. Она руководит Phill McKraken и Ima Little.

Phill McKraken не имеет подчиненных.

Ima Little менеджер. Она руководит Betsy Ross.
```

```
Betsy Ross не имеет подчиненных.

Jane Doe менеджер. Она руководит Walter H. Potter и Wendy B.K. McDonald.

Walter H. Potter не имеет подчиненных.

Wendy B.K. McDonald менеджер. Она руководит Craig F. Frye, Hardy
Hamburg, и Rich Shaker.

Craig F. Frye не имеет подчиненных.

Hardy Hamburg не имеет подчиненных.

Rich Shaker не имеет подчиненных.

Mike Rosenbaum менеджер. Он руководит Cindy Post-Kellog и Oscar A.
Winner.

Cindy Post-Kellog менеджер. Она руководит Allen Bran, Frank N. Berry, и
Jack Apple.

Allen Bran не имеет подчиненных.

Frank N. Berry не имеет подчиненных.

Jack Apple не имеет подчиненных.

Oscar A. Winner менеджер. Он руководит Jack Nicklaus, Tom Hanks, и Susan
Sarandon.

Jack Nicklaus менеджер. Он руководит R.P. McMurphy.

R.P. McMurphy не имеет подчиненных.

Tom Hanks менеджер. Он руководит Forrest Gump и Andrew Beckett.

Forrest Gump не имеет подчиненных.

Andrew Beckett не имеет подчиненных.

Susan Sarandon  менеджер. Она руководит Helen Prejean.

Helen Prejean не имеет подчиненных.
```

Более серьезное применение обхода в прямом порядке – преобразование дерева выражения в префиксную нотацию. В примере 5.5 приведен фрагмент

документа на языке MathML, а в примере 5.6 – программа его преобразования в Lisp-подобный синтаксис. В примере 5.7 представлен результат работы.

Пример 5.5. Фрагмент документа на языке MathML, представляющий уравнение $x^2 + 4x + 4 = 0$

```
<apply>
  <eq/>
  <apply>
    <plus/>
    <apply>
      <power/>
      <ci>x</ci>
      <cn>2</cn>
    </apply>
    <apply>
      <times/>
      <cn>4</cn>
      <ci>x</ci>
    </apply>
    <cn>4</cn>
  </apply>
  <cn>0</cn>
</apply>
```

Пример 5.5. Таблица стилей для преобразования фрагмента на языке MathML в префиксную нотацию

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

    <xsl:template match="apply">
       <xsl:value-of select="local-name(*[1])"/>
       <xsl:text>(</xsl:text>
       <xsl:apply-templates/>
       <xsl:text>)</xsl:text>
       <xsl:if test="following-sibling::*">,</xsl:if>
    </xsl:template>

    <xsl:template match="ci|cn">
       <xsl:value-of select="."/>
       <xsl:if test="following-sibling::*">,</xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

Пример 5.7. Результат

```
eq(plus(power(x,2),times(4,x),4),0)
```

Конвертер MathML – не самый чистый пример обхода в прямом порядке, поскольку в MathML математические выражения кодируются нетрадиционно. Но это безусловно пример идиомы «прямого порядка», так как первым обрабатывается элемент `apply` (и выводится имя первого дочернего элемента), а затем рекурсивно его потомки (с помощью элемента `<xsl:apply-templates/>`). Код, следующий за элементом `<xsl:apply-templates/>` балансирует скобки, вставляет в нужные места запятые, но не выполняет дополнительных рекурсивных вызовов.

5.5. Выполнение обхода в обратном порядке

Задача

Требуется рекурсивно обработать сначала потомков элемента, а затем его самого.

Решение

Решения в этом рецепте имеют такой общий вид:

```
<xsl:template match="node()">
    <!-- Обработать потомков -->
    <xsl:apply-templates/>

    <!-- Сделать что-то с текущим узлом -->

</xsl:template>
```

Обсуждение

Термин *обратный порядок* применяется, когда речь идет об обходе дерева таким образом, что сначала рекурсивно посещаются потомки корневого узла (в обратном порядке), а затем он сам. При таком обходе первыми мы обрабатываем листовые узлы, а затем поднимаемся вверх по дереву до корня.

Применив обход в обратном порядке в структурной диаграмме организации (*orgchart.xml*), мы можем напечатать информацию о том, кто кому подчиняется, начиная с нижнего уровня иерархии. Код программы приведен в примере 5.8, а в примере 5.9 показан результат ее работы.

Пример 5.8. Таблица стилей

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

  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>
```

```
<xsl:template match="/employee" priority="10">
  <xsl:apply-templates/>
  <xsl:value-of select="@name"/><xsl:text> глава компании. </xsl:text>
  <xsl:call-template name="reportsTo"/>
  <xsl:call-template name="HimHer"/> <xsl:text>. </xsl:text>
  <xsl:text>&#xa;&#xa;</xsl:text>
</xsl:template>

<xsl:template match="employee[employee]">
  <xsl:apply-templates/>
  <xsl:value-of select="@name"/><xsl:text> менеджер. </xsl:text>
  <xsl:call-template name="reportsTo"/>
  <xsl:call-template name="HimHer"/> <xsl:text>. </xsl:text>
  <xsl:text>&#xa;&#xa;</xsl:text>
</xsl:template>

<xsl:template match="employee">
  <xsl:text>Никто не подчиняется </xsl:text>
  <xsl:value-of select="@name"/><xsl:text>. &#xa;</xsl:text>
</xsl:template>

<xsl:template name="HimHer">
  <xsl:choose>
     <xsl:when test="@sex = 'male' ">
         <xsl:text>ему</xsl:text>
     </xsl:when>
     <xsl:otherwise>
         <xsl:text>ей</xsl:text>
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="reportsTo">
 <xsl:for-each select="*">
  <xsl:choose>
     <xsl:when test="position() &lt; last() - 1 and last() > 2">
        <xsl:value-of select="@name"/><xsl:text>, </xsl:text>
     </xsl:when>
     <xsl:when test="position() = last() - 1  and last() > 1">
        <xsl:value-of select="@name"/><xsl:text> и </xsl:text>
     </xsl:when>
     <xsl:when test="position() = last() and last() = 1">
        <xsl:value-of select="@name"/><xsl:text> подчиняется </xsl:text>
     </xsl:when>
```

```
        <xsl:when test="position() = last()">
           <xsl:value-of select="@name"/><xsl:text> подчиняются </xsl:text>
        </xsl:when>
        <xsl:otherwise>
           <xsl:value-of select="@name"/>
        </xsl:otherwise>
      </xsl:choose>
      </xsl:for-each>
   </xsl:template>

</xsl:stylesheet>
```

Пример 5.9. Результат

```
Никто не подчиняется Phill McKraken.
Никто не подчиняется Betsy Ross.
Ima Little менеджер. Betsy Ross подчиняется ей.

Nancy Pratt менеджер. Phill McKraken и Ima Little подчиняются ей.

Никто не подчиняется Walter H. Potter.
Никто не подчиняется Craig F. Frye.
Никто не подчиняется Hardy Hamburg.
Никто не подчиняется Rich Shaker.
Wendy B.K. McDonald менеджер. Craig F. Frye, Hardy Hamburg, и Rich
Shaker подчиняются ей.

Jany Doe менеджер. Walter H. Potter и Wendy B.K. McDonald подчиняются ей.

Никто не подчиняется Allen Bran.
Никто не подчиняется Frank N.Berry.
Никто не подчиняется Jack Apple.
Cindy Post-Kellog менеджер. Allen Bran, Frank N.Berry и Jack Apple
подчиняются ей.

Никто не подчиняется R.P. McMurphy.
Jack Nicklaus менеджер. R.P. McMurphy подчиняется ему.

Никто не подчиняется Forrest Gump.
Никто не подчиняется Andrew Beckett.
Tom Hanks менеджер. Forrest Gump и Andrew Beckett подчиняются ему.

Никто не подчиняется Helen Prejean.
Susan Sarandon менеджер. Helen Prejean подчиняется ей.

Oscar Winner менеджер. Jack Nicklaus, Tom Hanks, и Susan Sarandon подчи-
няются ему.
```

```
Mike Rosenbaum менеджер. Cindy Post-Kellog и Oscar Winner подчиняются ему.

Jil Michel глава компании. Nancy Pratt, Jane Doe, и Mike Rosenbaum
подчиняются ей.
```

5.6. Выполнение обхода во внутреннем порядке

Задача

Имеется XML-документ или фрагмент документа, представляющий выражение, которое требуется обработать во внутреннем порядке.

Решение

Обход во внутреннем порядке применяется как правило к двоичным деревьям. Общий вид алгоритма таков:

```
<xsl:template match="node()">
    <!-- Обработать левое поддерево -->
    <xsl:apply-templates select="*[1]"/>

    <!-- Сделать что-то с текущим узлом -->

    <!-- Обработать правое поддерево -->
    <xsl:apply-templates select="*[2]"/>
</xsl:template>
```

Однако обход во внутреннем порядке можно обобщить и на n-арные деревья следующим образом:

```
<xsl:template match="node()">
    <xsl:variable name="current-node" select="."/>
    <!-- Обработать левое поддерево -->
    <xsl:apply-templates select="*[1]"/>

    <!-- Сделать что-то с текущим узлом -->

    <!-- Рекурсивно применить к средним дочерним элементам -->
    <xsl:for-each select="*[position() > 1 and position() &lt; last()">

            <!-- Обработать "левое" поддерево -->
            <xsl:apply-templates select="."/>

            <!-- Сделать что-то с узлом $current-node -->

    </xsl:for-each>
```

```
    <!-- Обработать правое поддерево -->
    <xsl:apply-templates select="*[last()]"/>

</xsl:template>
```

Смысл этого алгоритма будет проще понять, если обратиться к рис. 5.1, на котором показан двоичный эквивалент n-арного дерева. Обобщенный обход n-арного дерева во внутреннем порядке дает тот же результат, что обход во внутреннем порядке эквивалентного ему двоичного дерева.

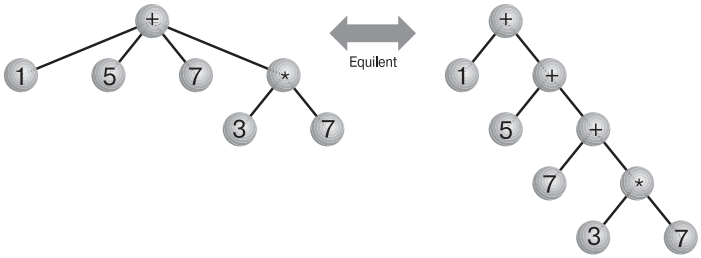


Рис. 5.1. Эквивалентность n-арного и двоичного дерева

Обсуждение

Этот вид обхода находит гораздо меньше применений, чем другие виды, рассмотренные в настоящей главе. Одно из значимых приложений приведено в примерах 5.10 и 5.11. Это программа преобразования MathML-разметки в инфиксное выражение на языке C или Java. Результат работы показан в примере 5.12.

Пример 5.10. Фрагмент документа на языке MathML

```
<apply>
    <eq/>
    <apply>
         <plus/>
         <apply>
               <minus/>
               <ci>y</ci>
               <cn>2</cn>
         </apply>
         <apply>
               <times/>
               <cn>4</cn>
               <apply>
                  <plus/>
                  <ci>x</ci>
                  <cn>1</cn>
               </apply>
```

```
        </apply>
        <cn>8</cn>
    </apply>
    <cn>0</cn>
</apply>
```

Пример 5.11. Обход фрагмента на языке MathML во внутреннем порядке, порождающий выражение на языке C

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

      <xsl:output method="text"/>
      <xsl:strip-space elements="*"/>

      <!-- Таблица для преобразования имен MathML-операций в операторы C -->
      <C:operator mathML="plus" c="+" precedence="2"/>
      <C:operator mathML="minus" c="-" precedence="2"/>
      <C:operator mathML="times" c="*" precedence="3"/>
      <C:operator mathML="div" c="/" precedence="3"/>
      <C:operator mathML="mod" c="%" precedence="3"/>
      <C:operator mathML="eq" c="==" precedence="1"/>

      <!-- Загрузить таблицу преобразования операций в переменную -->

      <xsl:variable name="ops" select="document('')/*/C:operator"/>

      <xsl:template match="apply">
         <xsl:param name="parent-precedence" select="0"/>

         <!-- Сопоставить MathML-операции знак и приоритет оператора -->
         <xsl:variable name="mathML-opName" select="local-name(*[1])"/>
         <xsl:variable name="c-opName"
           select="$ops[@mathML=$mathML-opName]/@c"/>
         <xsl:variable name="c-opPrecedence"
           select="$ops[@mathML=$mathML-opName]/@precedence"/>

         <!-- Если приоритет охватывающего выражения больше, чем приоритет
         текущего подвыражения, необходима скобка -->
         <xsl:if test="$parent-precedence > $c-opPrecedence">
           <xsl:text>(</xsl:text>
         </xsl:if>

         <!-- Рекурсивно обработать левое поддерево, которое находится
```

```
        в позиции 2, считая от элемента apply -->
        <xsl:apply-templates select="*[2]">
           <xsl:with-param name="parent-precedence"
               select="$c-opPrecedence"/>
           </xsl:apply-templates>

         <!-- Обработать текущий узел (т.е. оператор в позиции 1, считая
        от элемента apply -->
         <xsl:value-of select="concat(' ',$c-opName,' ')"/>

         <!-- Рекурсивно обработать средние дочерние элементы -->
         <xsl:for-each select="*[position()>2 and
                              position() &lt; last()]">
           <xsl:apply-templates select=".">
              <xsl:with-param name="parent-precedence"
                  select="$c-opPrecedence"/>
           </xsl:apply-templates>
           <xsl:value-of select="concat(' ',$c-opName,' ')"/>
          </xsl:for-each>

          <!-- Рекурсивно обработать правое поддерево -->
          <xsl:apply-templates select="*[last()]">
           <xsl:with-param name="parent-precedence"
                     select="$c-opPrecedence"/>
          </xsl:apply-templates>

          <!-- Если приоритет охватывающего выражения больше, чем приоритет
          текущего подвыражения, необходима скобка -->
          <xsl:if test="$parent-precedence > $c-opPrecedence">
           <xsl:text>)</xsl:text>
         </xsl:if>

    </xsl:template>

    <xsl:template match="ci|cn">
          <xsl:value-of select="."/>
    </xsl:template>

</xsl:stylesheet>
```

Пример 5.12. Результат

```
y — 2 + 4 * (x + 1) + 8 == 0
```

Понятно, что эту таблицу стилей нельзя назвать полноценным транслятором с языка MathML. Впрочем, в главе 9 мы обсудим эту проблему более детально.

5.7. Выполнение обхода по уровням

Задача

Требуется упорядочить элементы по возрастанию уровня (глубины в дереве). Другими словами, нужно обойти дерево в ширину.

Решение

Эта задача буквально создана для применения команды `xsl:for-each` совместно с `xsl:sort`.

```
<xsl:for-each select="//*">
    <xsl:sort select="count(ancestor::*)" data-type="number"/>
    <!-- Обработать текущий элемент -->
</xsl:for-each>
```

Рекурсивное решение длиннее и не так очевидно.

```
<xsl:template match="/*">
    <xsl:call-template name="level-order"/>
</xsl:template>

<xsl:template name="level-order">
<xsl:param name="max-level" select="10"/>
<xsl:param name="current-depth" select="1"/>

<xsl:choose>
    <xsl:when test="$current-depth &lt;= $max-level">
        <!-- обработать текущий уровень -->
        <xsl:call-template name="level-order-aux">
           <xsl:with-param name="level"
                      select="$current-level"/>
           <xsl:with-param name="actual-level"
                      select="$current-level"/>
           </xsl:call-template>
           <!-- обработать предыдущий уровень -->
           <xsl:call-template name="level-order">
              <xsl:with-param name="current-level"
                         select="$current-level + 1"/>
           </xsl:call-template>
    </xsl:when>
</xsl:choose>

</xsl:template>

<xsl:template name="level-order-aux">
```

```
    <xsl:param name="level" select="1"/>
    <xsl:param name="actual-level" select="1"/>
    <xsl:choose>
       <xsl:when test="$level = 1">
          <!-- Здесь обрабатывается текущий элемент -->
          <!-- $actual-level — номер текущего уровня -->
       </xsl:when>
       <xsl:otherwise>
          <!-- Рекурсивно переходим к следующему уровню для всех
          дочерних элементов -->
          <xsl:for-each select="*">
           <xsl:call-template name="level-order-aux">
                 <xsl:with-param name="level"
                      select="$level - 1"/>
                 <xsl:with-param name="actual-level"
                      select="$actual-level"/>
            </xsl:call-template>
           </xsl:for-each>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

В этом решении требуется, чтобы вы либо задали произвольную верхнюю границу глубины дерева, либо предварительно вычислили ее. Решить эту задачу можно, например, отыскав узел, у которого нет детей и который имеет больше предков, чем любой другой узел, не имеющий детей.

```
<xsl:param name="max-level">
    <xsl:for-each select="//*[not(*)]">
       <xsl:sort select="count(ancestor::*)" data-type="number"
           order="descending" />
       <xsl:if test="position() = 1">
          <xsl:value-of select="count(ancestor::*) + 1" />
       </xsl:if>
   </xsl:for-each>
</xsl-param>
```

Обсуждение

Необходимость обходить XML-документ по уровням возникает, пожалуй, не слишком часто. Группировка узлов по глубине или по расстоянию от корня плохо согласуется с естественным потоком управления в XSLT, который рассчитан на обход дерева в глубину. Это с очевидностью следует из того факта, что для обработки узлов по глубине или, что эквивалентно, по числу предков приходится использовать команду `xsl:sort`. Сложность рекурсивного решения также свидетельствует о неестественности этого процесса.

Тем не менее, такой вид обхода иногда оказывается полезным. Например, с его помощью можно обработать структурную диаграмму организации так, чтобы работники выводились по удаленности от главы компании. Соответствующий код приведен в примере 5.13, а результат его работы – в примере 5.14.

Пример 5.13. Обход файла orgchart.xml по уровням с использованием сортировки

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

<xsl:template match="/">
  <xsl:for-each select="//employee">
    <xsl:sort select="count(ancestor::*)" order="ascending"/>
    <xsl:variable name="level" select="count(ancestor::*)"/>
    <xsl:choose>
       <xsl:when test="$level = 0">
            <xsl:value-of select="@name"/>
            <xsl:text> глава компании.&#xA;</xsl:text>
       </xsl:when>
          <xsl:otherwise>
           <xsl:value-of select="@name"/>
          <xsl:text> имеет </xsl:text>
          <xsl:value-of select="$level"/>
          <xsl:text> начальников(а).&#xA;</xsl:text>
       </xsl:otherwise>
    </xsl:choose>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Пример 5.14. Результат

```
Jil Michel глава компании.
Nancy Pratt имеет 1 начальников(а).
JaneDoe имеет 1 начальников(а).
Mike Rosenbaum имеет 1 начальников(а).
Phil McKraken имеет 2 начальников(а).
Ima Little имеет 2 начальников(а).
Walter H. Potter имеет 2 начальников(а).
Wendy B.K. MacDonald имеет 2 начальников(а).
Cindy Post-Kellog имеет 2 начальников(а).
Oscar A. Winner имеет 2 начальников(а).
```

```
Betsy Ross имеет 3 начальников(а).
Craig F. Frye имеет 3 начальников(а).
Hardy Hamburg имеет 3 начальников(а).
Rich Shaker имеет 3 начальников(а).
Allen Bran имеет 3 начальников(а).
Frank N. Berry имеет 3 начальников(а).
Jack Apple имеет 3 начальников(а).
Jack Nicklaus имеет 3 начальников(а).
Tom Hanks имеет 3 начальников(а).
Susan Sarandon имеет 3 начальников(а).
R.P. McMurphy имеет 4 начальников(а).
Forrest Gump имеет 4 начальников(а).
Andrew Beckett имеет 4 начальников(а).
Helen Prejean имеет 4 начальников(а).
```

Один из плюсов рекурсивного решения состоит в том, что легко понять, когда происходит переход на следующий уровень. Этой информацией можно воспользоваться для форматирования результата, как показано в примерах 5.15 и 5.16.

Пример 5.15. Обход файла orgchart.xml по уровням с использованием рекурсии

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

<xsl:output method="text" version="1.0" encoding="UTF-8"/>

<xsl:strip-space elements="*"/>

<xsl:template match="/employee">
    <xsl:call-template name="level-order"/>
</xsl:template>

<xsl:template name="level-order">
<xsl:param name="max-depth" select="10"/>
<xsl:param name="current-depth" select="0"/>

<xsl:choose>
  <xsl:when test="$current-depth &lt;= $max-depth">
    <xsl:variable name="text">
       <xsl:call-template name="level-order-aux">
           <xsl:with-param name="level" select="$current-depth"/>
         <xsl:with-param name="actual-level" select="$current-depth"/>
       </xsl:call-template>
    </xsl:variable>
    <xsl:if test="normalize-space($text)">
        <xsl:value-of select="$text"/>
```

```
        <xsl:text>&#xa;</xsl:text>
        <xsl:call-template name="level-order">
           <xsl:with-param name="current-depth" select="$current-depth + 1"/>
           </xsl:call-template>
        </xsl:if>
    </xsl:when>
</xsl:choose>

</xsl:template>

<xsl:template name="level-order-aux">
    <xsl:param name="level" select="0"/>
    <xsl:param name="actual-level" select="0"/>
    <xsl:choose>
       <xsl:when test="$level = 0">
         <xsl:choose>
           <xsl:when test="$actual-level = 0">
          <xsl:value-of select="@name"/>
          <xsl:text> глава компании.&#xA;</xsl:text>
       </xsl:when>
       <xsl:otherwise>
          <xsl:value-of select="@name"/>
          <xsl:text> имеет </xsl:text>
          <xsl:value-of select="$actual-level"/>
          <xsl:text> начальников(а).&#xA;</xsl:text>
       </xsl:otherwise>
     </xsl:choose>
    </xsl:when>
    <xsl:otherwise>
       <xsl:for-each select="employee">
          <xsl:call-template name="level-order-aux">
           <xsl:with-param name="level" select="$level - 1"/>
           <xsl:with-param name="actual-level" select="$actual-level"/>
          </xsl:call-template>
       </xsl:for-each>
     </xsl:otherwise>
   </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 5.16. Результат

```
Jil Michel глава компании.

Nancy Pratt имеет 1 начальников(а).
```

```
JaneDoe имеет 1 начальников(а).
Mike Rosenbaum имеет 1 начальников(а).

Phil McKraken имеет 2 начальников(а).
Ima Little имеет 2 начальников(а).
Walter H. Potter имеет 2 начальников(а).
Wendy B.K. MacDonald имеет 2 начальников(а).
Cindy Post-Kellog имеет 2 начальников(а).
Oscar A. Winner имеет 2 начальников(а).

Betsy Ross имеет 3 начальников(а).
Craig F. Frye имеет 3 начальников(а).
Hardy Hamburg имеет 3 начальников(а).
Rich Shaker имеет 3 начальников(а).
Allen Bran имеет 3 начальников(а).
Frank N. Berry имеет 3 начальников(а).
Jack Apple имеет 3 начальников(а).
Jack Nicklaus имеет 3 начальников(а).
Tom Hanks имеет 3 начальников(а).
Susan Sarandon имеет 3 начальников(а).

R.P. McMurphy имеет 4 начальников(а).
Forrest Gump имеет 4 начальников(а).
Andrew Beckett имеет 4 начальников(а).
Helen Prejean имеет 4 начальников(а).
```

Если вам почему-то захочется посещать узлы на одном из уровней в случайном порядке, то можно следующим образом определить ключ:

```
<xsl:key name="level" match="employee" use="count(ancestor::*)"/>

<xsl:template match="/">
    <xsl:for-each select="key('level', 3)">
         <!-- сделать что-то с узлами на уровне 3 -->
    </xsl:for-each>
</xsl:template>
```

Можно также уточнить условие сопоставления или использовать предикат совместно с функцией `key`:

```
<xsl:key name="level" match="employee" use="count(ancestor::*)"/>

<xsl:template match="/">
  <xsl:for-each select="key('level', 3)[@sex='female']">
     <!-- сделать что-то с работниками-женщинами на уровне 3 -->
    </xsl:for-each>
</xsl:template>
```

Прибегать к механизму определения ключей в XSLT необязательно, поскольку всегда можно написать `select="//*[count(ancestor::*)=3]"`, если нужно получить все элементы на уровне `level-3`. Однако, если в таблице стилей будет многократно вычисляться такое выражение, то упадет производительность. Вообще, если исходный файл имеет четко определенную структуру, то гораздо эффективнее явно перейти на нужный уровень, например, с помощью выражения `select="/employee/employee/employee"`, хотя подобная навигация может оказаться довольно громоздкой.

5.8. Обработка узлов по позиции

Задача

Требуется обработать узлы в порядке, который является функцией от их позиции в документе или в наборе узлов.

Решение

Примените команду `xsl:sort`, указав в качестве атрибута `select` функцию `position()` или `last()`. Простейший пример такого рода – обработка узлов в обратном порядке документа.

```
<xsl:apply-templates>
   <xsl:sort select="position()" order="descending" data-type="number"/>
</xsl:apply-templates>
```

или

```
<xsl:for-each select="*">
   <xsl:sort select="position()" order="descending" data-type="number"/>
   <!-- ... -->
</xsl:for-each>
```

Еще одно популярное применение этого приема – обход набора узлов так, как если бы он представлял собой матрицу с заданным числом колонок:

```
<xsl:for-each select="*">
   <xsl:sort select="(position() - 1)" mod 3"/>
   <!-- ... -->
</xsl:for-each>
```

Или даже более элегантно:

```
<xsl:for-each select="*[position() mod 3 = 1]">
   <xsl:apply-templates
       select=". | following-sibling::*[position() &lt; 3]" />
</xsl:for-each>
```

Иногда приходится использовать функцию `position()`, чтобы отделить первый узел в наборе от всех остальных. Это позволяет выполнять сложные операции

агрегирования над документом с помощью рекурсии. Я называю такой прием *рекурсивное агрегирование*. В общем виде он записывается так:

```
<xsl:template name="aggregation">
  <xsl:param name="node-set"/>
  <xsl:choose>
    <xsl:when test="$node-set">
       <!-- Вычисляем некоторую функцию от первого элемента, возвращающую
       значение,по которому мы хотим агрегировать. Функция может
       зависеть от типа элемента (т.е. быть полиморфной) -->
       <xsl:variable name="first">
          <xsl:apply-templates select="$node-set[1]" mode='calc'/>
       </xsl:variable>
       <!-- Рекурсивно обрабатываем остальные узлы, пользуясь position() -->
       <xsl:variable name="rest">
          <xsl:call-template name="aggregation">
           <xsl:param name="node-set"/>
               select="$node-set[position()!=1]"/>
           </xsl:template>
       </xsl:variable>
       <!-- Выполняем какую-то операцию агрегирования. Возможно, вызывать
       шаблон для этого и не потребуется. Например, это может быть нечто такое:
            $first + $rest или
            $first * $rest или
            concat($first, $rest) и т.д. -->
       <xsl:call-template name="aggregate-func">
          <xsl:with-param name="a" select="$first"/>
          <xsl:with-param name="b" select="$rest"/>
       </xsl:call-template>
    </xsl:when>
    <!-- Ниже IDENTITY-VALUE следует заменить идентификатором агрегирующей
    функции aggregate-func. Например, 0 - идентификатор функции сложения,
    1 - идентификатор функции умножения, "" - идентификатор функции
    конкатенации и т.д. -->
    <xsl:otherwise>IDENTITY-VALUE</xsl:otherwise>
</xsl:template>
```

Обсуждение

Для XSLT естественно обрабатывать узлы в порядке документа или – что то же самое – в порядке позиций. Таким образом, следующие два фрагмента эквивалентны (сортировка избыточна):

```
<xsl:for-each select="*">
    <xsl:sort select="position()"/>
```

```
    <!-- ... -->
</xsl:for-each>

<xsl:for-each select="*">
    <!-- ... -->
</xsl:for-each>
```

Применив вариацию этой идеи, можно распечатать структурную диаграмму организации в две колонки, как показано в примерах 5.17 и 5.18.

Пример 5.17. columns-orgchart.xslt

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

<xsl:output method="text" />
<xsl:strip-space elements="*"/>

<xsl:template match="employee[employee]">
<xsl:value-of select="@name"/>
<xsl:text>&#xA;</xsl:text>
<xsl:call-template name="dup">
  <xsl:with-param name="input" select=" '-' "/>
  <xsl:with-param name="count" select="80"/>
</xsl:call-template>
<xsl:text>&#xA;</xsl:text>
<xsl:for-each select="employee[(position() - 1) mod 2 = 0]">
  <xsl:value-of select="@name"/>
  <xsl:call-template name="dup">
    <xsl:with-param name="input" select=" ' ' "/>
    <xsl:with-param name="count" select="40 - string-length(@name)"/>
  </xsl:call-template>
  <xsl:value-of select="following-sibling::*[1]/@name"/>
  <xsl:text>&#xA;</xsl:text>
</xsl:for-each>
<xsl:text>&#xA;</xsl:text>
<xsl:apply-templates/>
</xsl:template>

<xsl:template name="dup">
<xsl:param name="input"/>
<xsl:param name="count" select="1"/>
<xsl:choose>
  <xsl:when test="not($count) or not($input)"/>
```

```
  <xsl:when test="$count = 1">
     <xsl:value-of select="$input"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:if test="$count mod 2">
       <xsl:value-of select="$input"/>
    </xsl:if>
    <xsl:call-template name="dup">
       <xsl:with-param name="input"
            select="concat($input,$input)"/>
       <xsl:with-param name="count"
            select="floor($count div 2)"/>
    </xsl:call-template>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 5.18. Результат

```
Jil Michel
-----------------------------------------
Nancy Pratt               JaneDoe
Mike Rosenbaum

Nancy Pratt
-----------------------------------------
Phil McKraken                    Ima Little

Ima Little
-----------------------------------------
Betsy Ross

Jane Doe
-----------------------------------------
Walter H. Potter          Wendy B.K. MacDonald

Wendy B.K. MacDonald
-----------------------------------------
Craig F. Frye                    Hardy Hamburg
Rich Shaker

Mike Rosenbaum
-----------------------------------------
```

```
Cindy Post-Kellog         Oscar A. Winner

Cindy Post-Kellog
----------------------------------------
Allen Bran                Frank N. Berry
Jack Apple

Oscar A. Winner
----------------------------------------
Jack Nicklaus                    Tom Hanks
Susan Sarandon

Jack Nicklaus
----------------------------------------
R.P. McMurphy

Tom Hanks
----------------------------------------
Forrest Gump                     Andrew Beckett

Susan Sarandon
----------------------------------------
Helen Prejean
```

Один из примеров рекурсивного агрегирования – это таблица стилей, которая вычисляет общую сумму комиссионных, выплачиваемых продавцам. Предполагается, что сумма комиссионных – функция от суммы продаж всех продуктов. Код и результат приведены в примерах 5.19 и 5.20.

Пример 5.19. total-commision.xslt

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

<xsl:template match="salesBySalesperson">
  <xsl:text>Общая сумма комиссионных = </xsl:text>
  <xsl:call-template name="total-commision">
    <xsl:with-param name="salespeople" select="*"/>
  </xsl:call-template>
</xsl:template>

<!-- По умолчанию продавец получает 2% от суммы сделки без основной
зарплаты -->
<xsl:template match="salesperson" mode="commision">
  <xsl:value-of select="0.02 * sum(product/@totalSales)"/>
```

```
</xsl:template>

<!-- Продавцы разряда > 4 получают $10000.00 + комиссионные в размере 0.5%  -->
<xsl:template match="salesperson[@seniority > 4]"
              mode="commision" priority="1">
  <xsl:value-of select="10000.00 + 0.05 * sum(product/@totalSales)"/>
</xsl:template>

<!-- Продавцы разряда > 8 получают (разряд * $2000.00) + комиссионные в
размере 0.8%  -->
<xsl:template match="salesperson[@seniority > 8]"
               mode="commision" priority="2">
  <xsl:value-of select="@seniority * 2000.00 + 0.08 *
       sum(product/@totalSales)"/>
</xsl:template>

<xsl:template name="total-commision">
  <xsl:param name="salespeople"/>
  <xsl:choose>
     <xsl:when test="$salespeople">
     <xsl:variable name="first">
     <xsl:apply-templates select="$salespeople[1]" mode="commision"/>
     </xsl:variable>
     <xsl:variable name="rest">
     <xsl:call-template name="total-commision">
       <xsl:with-param name="salespeople"
           select="$salespeople[position()!=1]"/>
      </xsl:call-template>
      </xsl:variable>
     <xsl:value-of select="$first + $rest"/>
    </xsl:when>
    <xsl:otherwise>0</xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 5.20. Результат

```
Общая сумма комиссионных = 471315
```

XSLT 2.0

В версии 2.0 для вычисления суммы комиссионных можно комбинировать функции и шаблоны, чтобы избежать рекурсии, но вместе с тем в полной мере задействовать свойственный шаблонам механизм сопоставления с образцами.

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2004/10/xpath-functions"
    xmlns:ckbk="http://www.oreilly.com/catalog/xsltckbk">

<xsl:output method="text"/>

<xsl:template match="salesBySalesperson">
  <xsl:text>Общая сумма комиссионных = </xsl:text>
  <xsl:value-of select="ckbk:total-commision(*)"/>
</xsl:template>

<!-- По умолчанию продавец получает 2% от суммы сделки без основной
зарплаты -->
<xsl:template match="salesperson" mode="commision" as="xs:double">
   <xsl:sequence select="0.02 * sum(product/@totalSales)"/>
</xsl:template>

<!-- Продавцы разряда > 4 получают $10000.00 + комиссионные в размере
0.5%  -->
<xsl:template match="salesperson[@seniority > 4]" mode="commision"
priority="1" as="xs:double">
  <xsl:sequence select="10000.00 + 0.05 * sum(product/@totalSales)"/>
</xsl:template>

<!-- Продавцы разряда > 8 получают (разряд * $2000.00) + комиссионные
в размере 0.8%  -->
<xsl:template match="salesperson[@seniority > 8]" mode="commision"
priority="2" as="xs:double">
  <xsl:sequence select="@seniority * 2000.00 + 0.08 *
        sum(product/@totalSales)"/>
</xsl:template>

<xsl:function name="ckbk:total-commision" as="xs:double">
  <xsl:param name="salespeople" as="node()*"/>
  <xsl:sequence select="sum(for $s in $salespeople return
ckbk:commision($s))"/>
</xsl:function>

<xsl:function name="ckbk:commision" as="xs:double">
  <xsl:param name="salesperson" as="node()"/>
  <xsl:apply-templates select="$salesperson" mode="commision"/>
</xsl:function>

</xsl:stylesheet>
```

См. также

Майкл Кэй приводит изящный пример рекурсивного агрегирования на стр. 535 своей книги *XSLT Programmer's Reference* (Wrox Press, 2001). В нем вычисляется общая площадь различных фигур, причем формула вычисления площади зависит от типа фигуры.

Джени Теннисон также приводит примеры рекурсивного агрегирования и альтернативные способы решения аналогичных задач в книге *XSLT and XPath on the Edge* (M&T Books, 2001).

Глава 6. XSLT 2.0

> Если единственное, что мы должны предлагать, – это улучшенный вариант прошлого, то сегодня может быть только хуже, чем вчера. Загипнотизированные образами прошлого, мы рискуем утратить способность к творческим изменениям.
>
> *Роберт Хьюисон*

6.0. Введение

В XSLT 2.0 внесено множество дополнений и улучшений, с помощью которых решать трудные задачи стало проще. Эта глава призвана помочь ветеранам XSLT 1.0 перейти на версию 2.0, а новичкам – разобраться в том, как лучше проектировать таблицы стилей в этой версии.

Значительная часть новой функциональности XSLT 2.0 проистекает из изменений в XPath 2.0, поэтому, если вы пропустили главу 1, сейчас самое время прочитать ее.

Следуя общей тенденции развития программных технологий, версия 2.0 представляет собой усовершенствованную предыдущую версию, а не заставляет полностью изменить подход к написанию таблиц стилей. Ей недостает некоторых черт, которые могли бы сделать язык еще лучше (например, интроспекции и прямой поддержки функций более высокого порядка). Тем не менее, версия XSLT 2.0 далеко ушла по пути облегчения разработки сложных таблиц стилей. Основные особенности 2.0 – это функции в XPath, группировка, расширенные режимы, улучшенные идиомы повторного использования кода, более развитая система типов и дополнительная поддержка в области обработки текста. Все это, конечно, используется в рецептах для XSLT 2.0, содержащихся в этой книге, но эта глава может служить справочником по новым возможностям как таковым.

6.1. Преобразование простых именованных шаблонов в функции XSLT

Задача

В XSLT 1.0 не было поддержки для написания функций, а именованные шаблоны – не слишком удачная замена.

Решение

Если нужно всего лишь вычислить результат, а не породить новое сериализованное содержимое, отдавайте предпочтение функциям XSLT 2.0. Ниже приведены разнородные примеры ситуаций, когда функции гораздо удобнее именованных шаблонов:

```
<!-- Математические вычисления -->
<xsl:function name="ckbk:factorial" as="xs:decimal">
    <xsl:param name="n" as="xs:integer"/>
    <xsl:sequence select="if ($n eq 0) then 1
                          else $n * ckbk:factorial($n - 1)"/>
</xsl:function>


<!-- Простые отображения -->
<xsl:function name="ckbk:decodeColor" as="xs:string">
    <xsl:param name="colorCode" as="xs:integer"/>
    <xsl:variable name="colorLookup"
                  select="('black','red','orange','yellow',

'green','blue','indigo','violet','white)"/>
    <xsl:sequence select="if ($colorCode ge 0 and
                              $colorCode lt count($colorLookup))
                          then $colorLookup[$colorCode - 1]
                          else 'no color'"/>
</xsl:function>

<!-- Манипуляции со строками -->

<xsl:function name="ckbk:reverse">
    <xsl:param name="input" as="xs:string"/>
    <xsl:sequence select="codepoints-to-string(reverse(
                         string-to-codepoints($input)))"/>
</xsl:function>
```

Обсуждение

Напомним, что именованные шаблоны – альтернатива шаблонам, вызываемым в результате успешного сопоставления с образцом. Их можно уподобить процедурам в традиционных языках, поскольку программист явно передает управление именованному шаблону командой `xsl:call-template`, а не полагается на декларативную семантику сопоставления. Приятной особенностью XSLT (в обеих версиях) является тот факт, что один и тот же шаблон можно вызывать как в результате сопоставления, так и по имени.

Определяемые пользователем функции в XSLT 2.0 – не замена именованным шаблонам. Главный вопрос, который нужно задать себе, выбирая то или другое, звучит так: «Я собираюсь просто вычислить результат или создать повторно используемый генератор содержимого?» В первом случае лучше воспользоваться функцией, а во втором – шаблоном. В книге *XSLT Programmer's Reference* Майкл Кэй рекомендует использовать функции, когда вы просто отбираете узлы, а шаблоны – когда создаете новые узлы, несмотря даже на то, что XSLT допускает применение функций и во втором случае.

Следующая функция отбирает узлы:

```
<xsl:function name="getParts" as="item()*">
    <xsl:param name="startPartId" as="xs:string"/>
    <xsl:param name="endPartId" as="xs:string"/>
    <xsl:sequence select="//Parts/part[@partId ge $startPartId
                                       and @partId le $endPartId]"/>
</xsl:function>
```

А эта функция создает новые узлы, но шаблон, пожалуй, был бы здесь уместнее:

```
<xsl:function name="getPartsElem" as="item()">
    <xsl:param name="startPartId" as="xs:string"/>
    <xsl:param name="endPartId" as="xs:string"/>
    <Parts>
           <xsl:copy-of select="//Parts/part[@partId ge $startPartId
                                          and @partId le $endPartId]"/>
    </Parts>
</xsl:function>
```

6.2. Для группировки пользуйтесь командой for-each-group, а не методом Мюнха

Задача

В XSLT 1.0 нет явной поддержки для группировки, поэтому приходится изобретать косвенные и не слишком понятные методы.

Решение

Всякий раз, когда нужна группировка, пользуйтесь преимуществами весьма мощной команды `xsl:for-each-group`. У нее есть обязательный атрибут `select`, значением которого является выражение, отбирающее множество узлов, которые нужно сгруппировать. Чтобы определить критерии разбиения этого множества на группы, задается один из четырех атрибутов группировки. Все они будут рассмотрены ниже. По ходу обработки вы можете пользоваться функцией `current-group()` для доступа ко всем узлам в текущей группе. Функция `current-grouping-key()` возвращает значение ключа, определяющего текущую группу, в случае, когда

группировка осуществляется по значениям или по соседним узлам. Для группировки по начальному или конечному узлу эта функция не возвращает никакого значения.

Группы можно сортировать, если вставить одну или несколько команд `xsl:sort`, определяющих критерии сортировки. Собственно, то же самое делается и при использовании команды `for-each`.

Группировка по значениям (group-by="expression")

Классическая задача группировки часто возникает при генерировании отчетов. Рассмотрим данные о продажах. Менеджер по продажам часто хочет сгруппировать данные по регионам, по типам продукции или по продавцам в зависимости от стоящей перед ним задачи. Чтобы указать, по каким значениям группируются отобранные узлы, воспользуйтесь атрибутом `group-by`. Например, `group-by="@dept"` объединяет в одну группу узлы с одним и тем же значением атрибута `dept`.

```
<xsl:template match="Employees">
  <EmployeesByDept>
    <xsl:for-each-group select="employee" group-by="@dept">
       <dept name="{current-grouping-key()}">
           <xsl:copy-of select="current-group"/>
       </dept>
    </xsl:for-each-group>
  </EmployeesByDept>
</xsl:template>
```

Группировка по соседним узлам (group-adjacent="expression")

В некоторых случаях, в частности, при обработке документов требуется рассматривать узлы, обладающие каким-то общим значением, при условии, что они также являются соседними. Как и `group-by`, атрибут `group-adjacent` определяет значение, по которому осуществляется группировка, однако два узла с таким значением будут входить в одну группу лишь, если они соседние. Значение атрибута `group-adjacent` должно быть последовательностью, содержащей в точности один элемент; пустая или многоэлементная последовательность приводит к ошибке.

Рассмотрим документ, состоящий из элементов `para`, перемежающихся с элементами `heading`. Хотелось бы извлекать только элементы `para`, не теряя однако информацию о том, что последовательность таких элементов принадлежит одному и тому же разделу:

```
<xsl:template match="doc">
  <xsl:copy>
    <xsl:for-each-group select="*" group-adjacent="name()">
       <xsl:if test="self::para">
          <topic>
           <xsl:copy-of select="current-group()"/>
```

```
            </topic>
          </xsl:if>
        </xsl:for-each-group>
      </xsl:copy>
    </xsl:template>
```

Группировка по начальному узлу (group-starting-with="pattern")

Часто, особенно при обработке документов, группа взаимосвязанных узлов помечается специальным узлом, например title или subtitle, или каким-нибудь другим, обозначающим заголовок. Группировка по начальному узлу упрощает обработку таких слабо структурированных документов. Атрибут `group-starting-with` определяет образец, соответствующий узлам, которые отмечают начало каждой группы. Здесь есть прямая аналогия с атрибутом `match` в команде `xsl:template`. Если образец соответствует какому-то узлу в отобранном множестве, то в группу помещаются все следующие за ним узлы вплоть до следующего соответствия. Первый узел в наборе начинает группу вне зависимости от того, соответствует он образцу или нет. Это гарантирует, что в множестве имеется по меньшей мере одна группа, даже если ни один узел не соответствует образцу.

Классический пример – наделение структурой неструктурированного документа. XHTML может служить примером слабо структурированного языка разметки, особенно в части заголовочных элементов (`h1`, `h2` и т.д.). Следующее преобразование внесет в него некую структуру путем помещения внутрь элемента `div` каждой группы, начинающейся с элемента `h1`:

```
<xsl:template match="body">
  <xsl:copy>
   <xsl:for-each-group select="*" group-starting-with="h1">
    <div>
       <xsl:apply-templates select="current-group()"/>
    </div>
   </xsl:for-each-group>
  </xsl:copy>
</xsl:template>
```

Группировка по конечному узлу (group-ending-with="pattern")

Этот вид группировки аналогичен `group-starting-with`, но образец задается атрибутом `group-ending-with` и определяет последний узел в текущей группе. Первый узел в отобранном множестве начинает новую группу, поэтому по меньшей мере одна группа всегда существует, даже если нет соответствующих образцу узлов.

Из всех способов группировки у этого, вероятно, меньше всего применений. И объясняется это тем, что в документах, ориентированных на человека, для формирования групп обычно используются начальные, а не конечные элементы. В книге *XSLT Programmer's Reference* Майкл Кэй приводит пример серии документов, разбитых на части для передачи по каналам связи. Границы документов опознаются по отсутствию атрибута `continued='yes'`. Чуть более реалистичный

пример – структурирование плоского файла путем объединения в один кусок элементов на основе некоторого критерия, определяющего конец куска. Например, следующий код группирует абзацы в разделы по пять абзацев в каждом:

```
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="doc">
    <xsl:copy>
      <xsl:for-each-group select="para"
                          group-ending-with="para[position() mod 5 eq 0]">
        <section>
          <xsl:for-each select="current-group()">
            <xsl:copy>
                <xsl:apply-templates select="@*|node()"/>
            </xsl:copy>
          </xsl:for-each>
        </section>
      </xsl:for-each-group>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@* | node()">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Обсуждение

Метод Мюнха, названный в честь Стива Мюнха (Steve Muench) из корпорации Oracle, когда-то был новаторским способом группировки данных в XSLT 1.0. В нем используется встроенная в XSLT возможность индексировать документы по ключу. Смысл трюка в том, чтобы с помощью индекса эффективно получить множество уникальных ключей группировки, а затем воспользоваться этим множеством для обработки всех узлов в группе:

```
<xsl:key name="products-by-category" select="product" use="@category"/>

<xsl:template match="/">

    <xsl:for-each select="//product[count(. | key('products-by-category',
@category)[1]) = 1]">
    <xsl:variable name="current-grouping-key"
```

```
                select="@category"/>
        <xsl:variable name="current-group"
                      select="key('current-grouping-key',
                      $current-grouping-key)"/>
        <xsl:for-each select="$current-group/*">
            <!-- обработка элементов в группе -->
            <!-- при необходимости можно добавить и xsl:sort -->
        </xsl:for-each/>
    </xsl:for-each/>

<xsl:template match="/">
```

Хотя метод Мюнха будет работать и в версии 2.0, ему следует предпочесть команду `for-each-group`, так как она, скорее всего, не менее, а, возможно, даже более эффективна. Немаловажно также, что в этом случае код становится намного понятнее, особенно для неопытных пользователей.

К тому же, одна и та же конструкция позволяет выполнить четыре разных задачи группировки, тогда как метод Мюнха годится только для группировки по значениям. Пожалуй, единственная причина продолжать применять метод Мюнха в XSLT 2.0 – обратная совместимость с первой версией.

6.3. Модульность и режимы

Задача

Имеющиеся в XSLT 1.0 ограничения на использование режимов (атрибута mode) часто приводят к дублированию кода или лишней работе.

Решение

Пользуйтесь новыми, появившимися в XSLT 2.0 возможностями атрибута `mode` для устранения дублирования кода. Рассмотрим простую таблицу стилей, в которой один и тот же документ обрабатывается в два прохода с разными режимами. На каждом проходе по умолчанию текстовые узлы должны игнорироваться. В XSLT 1.0 пришлось бы написать что-то вроде:

```
<xsl:template match="text()" mode="mode1"/>
<xsl:template match="text()" mode="mode2"/>
```

А в версии 2.0 избыточность можно исключить:

```
<xsl:template match="text()" mode="mode1 mode2"/>
```

Или, если вы хотите обрабатывать все узлы:

```
<xsl:template match="text()" mode="#all"/>
```

Конечно, здесь выигрыш невелик, но он становится куда заметнее в более сложных таблицах стилей, где используется много режимов, или велик объем общего кода в разных режимах, или часто приходится что-то изменять.

Обсуждение

Пользуясь режимами в версии 2.0, я всегда стараюсь употреблять значение `#current` вместо явно поименованного режима, если требуется продолжать обработку в текущем режиме:

```
<xsl:template match="author" mode="index">
    <div class="author">
      <xsl:apply-templates mode="#current"/>
    </div>
</xsl:template>
```

У такого решения два очевидных достоинства. Во-первых, если позже вы сочтете, что режим назван неудачно, то для изменения имени не нужно будет модифицировать каждое из обращений к `xsl:apply-templates`. Во-вторых, если понадобится добавить в шаблон новые режимы, все будет работать без каких бы то ни было изменений:

```
<xsl:template match="author" mode="index body">
    <div class="author">
           <xsl:apply-templates mode="#current"/>
    </div>
</xsl:template>
```

6.4. Применение типов в целях безопасности и точности выражения намерений

Задача

Ограниченные средства контроля типов в XSLT 1.0 не дают написать по-настоящему устойчивый к ошибкам шаблон.

Решение

Пользуйтесь расширенной системой типов в XSLT 2.0 для создания безопасных относительно типов функций и шаблонов.

Задавайте атрибут as для элементов, содержащих или возвращающих данные

К таким элементам относятся `xsl:function`, `xsl:param`, `xsl:template`, `xsl:variable` и `xsl:with-param`.

Задавайте атрибут type для элементов, создающих данные

К таким элементам относятся `xsl:attribute`, `xsl:copy`, `xsl:copy-of`, `xsl:document`, `xsl:element` и `xsl:result-document`.

Обсуждение

Все совместимые со спецификацией XSLT 2.0 процессоры позволяют применять простые типы данных, к примеру `xs:integer` или `xs:string`, для описания переменных и параметров. Совместно с типами можно также употреблять символы *, + и ? для описания последовательностей:

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" >

<!-- x - последовательность из 0 или более строк -->
<xsl:variable name="x" select="para" as="xs:string*"/>

<!-- y - последовательность из 1 или более строк. Атрибут select написан
так, что это ограничение гарантированно выполняется, хотя, если вы точно
знаете, что должен быть по меньшей мере один элемент para, то это
выражение можно и опустить -->
<xsl:variable name="y"
              select="if (para) then para else ''"
              as="xs:string+"/>

<!-- z - последовательность из 0 или 1 строки.  -->
<xsl:variable name="z" select="para[1]" as="xs:string?"/>

</xsl:stylesheet>
```

Если процессор поддерживает схемы, то можно пойти дальше и ссылаться на простые и составные типы, описанные в схеме, определенной пользователем. Такому процессору информация о пользовательских типах должна передаваться с помощью команды верхнего уровня `xsl:import-schema`.

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:my="http://www.mydomain.com/ns/my">

<xsl:import-schema schema-location="my-schema.xsd"/>

<xsl:template match="/">
  <!-- Проверять, что результирующий элемент имеет тип my:BookType -->
  <xsl:element name="Book" type="my:BookType">
     <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

```
<!-- ... -->

</xsl:stylesheet>
```

Нельзя включать команду `xsl:import-schema`, если ваш процессор не поддерживает схемы. Если схемы все-таки поддерживаются, но вы хотите, чтобы ваши таблицы стилей были совместимы и с процессорами, не поддерживающими схемы, следует использовать атрибут `use-when="system-property('xsl:schema-aware') = 'yes'"` для всех элементов, требующих наличия процессора с поддержкой схем.

6.5. Избегайте подводных камней при переносе с XSLT 1.0 на 2.0

Задача

Не каждая таблица стилей, созданная для версии 1.0, будет без изменений работать и в 2.0.

Решение

При переносе таблиц стилей из версии 1.0 в 2.0 нужно помнить о нескольких подводных камнях. Часть из них можно обойти, включив режим совместимости с XSLT 1.0; для этого нужно задать атрибут version="1.0" в элементе stylesheet: `<xsl:stylesheet version="1.0">`. Но уж раз вы начали перенос старых программ в версию 2.0, то есть и другие способы справиться с несовместимостями.

Процессоры XSLT 2.0 не обязаны поддерживать режим обратной совместимости, хотя большинство все-таки поддерживает. Если это не так, процессор известит об ошибке.

Последовательности не преобразуются неявно в атомарные значения

Рассмотрим такой фрагмент: `<xsl:value-of select="substring-before(Name, ' ')"/>`. Что произойдет, если он вычисляется в контексте, где есть более одного элемента `Name`? В XSLT 1.0 используется только первый элемент, а остальные игнорируются. Но XSLT 2.0 в этом отношении ведет себя строже и извещает об ошибке типизации, так как первый аргумент функции `substring-before` должен быть последовательностью, содержащей 0 или 1 строку.

Чтобы не сталкиваться с такой ошибкой, возьмите за правило писать `<xsl:value-of select="substring-before(Name[1], ' ')"/>`. С другой стороны, иногда полезно получать извещение об этой ситуации, поскольку она может свидетельствовать о наличии ошибки в таблице стилей или входном документе. Альтернативное решение заключается в том, чтобы объединить несколько узлов в один перед тем, как подавать последовательность на вход функции, ожидающей единственный узел. Например, если написать `<xsl:value-of-select="substring-before(string-join(Name), ' ')"/>`, то ошибки не возникнет.

Типы не преобразуются неявно в другие типы

В XSLT 1.0 допускается большая свобода во всем, что касалось преобразований типов. Если функция ожидает число, а вы передали ей строку, то она сделает все возможное, чтобы преобразовать строку в число или наоборот. То же справедливо в отношении булевского значения и целого числа или строки и булевского значения. Старое поведение можно сохранить, установив режим совместимости с версией 1.0. Но можно также преобразовывать типы явно:

```
<xsl:variable name="a" select=" '10' "/>
<xsl:value-of select="sum($a, 1)"/> <!-- Ошибка -->
<xsl:value-of select="sum(number($a), 1)"/> <!-- Правильно -->

<xsl:value-of select="string-length(10)"/> <!-- Ошибка -->
<xsl:value-of select="string-length(string(10))"/> <!-- Правильно -->
<xsl:value-of select="string-length(string(true()))"/>
                                          <!-- Правильно, равно 4 -->

<xsl:value-of select="1 + true()"/> <!-- Ошибка -->
<xsl:value-of select="1 + number(true())"/> <!-- Правильно, равно 2 -->
```

Лишние параметры не игнорируются

В XSLT 1.0, если шаблону, вызываемому командой `xsl:call-template`, передавались параметры, которые в шаблоне не были определены, то они просто игнорировались. В версии 2.0 это считается ошибкой. Подавить такую ошибку можно, включив режим совместимости. Никаких других обходных путей не существует, можно лишь удалить лишние параметры или определить в существующем шаблоне значения параметров по умолчанию.

Более строгий контроль семантики приводит к ошибкам в сомнительных случаях

Это можно наблюдать на примере команд `xsl:number` и `xsl:template`. Если в `xsl:number` одновременно заданы атрибуты `level` и `value`, то в XSLT 1.0 `level` игнорируется, а в 2.0 это ошибка. Аналогично, в команде `xsl:template` в версии 2.0 запрещено указывать атрибут `priority` или `mode`, если отсутствует атрибут `match`.

Обсуждение

Включение режима совместимости для исправления ошибок в таблицах стилей, составленных для версии 1.0, имеет и другие последствия. В частности, в некоторых отношениях программа начинает вести себя по-другому. Точнее, в режиме совместимости:

- команда `xsl:value-of` возвращает только первый элемент последовательности, а не все элементы, разделенные пробелами;
- выражение, употребленное в качестве значения атрибута (например, `<foo values="{foo}"/>`), выводит только первый элемент последовательности, а не все элементы, разделенные пробелами;

- при использовании команды `xsl:number` выводится только первое число, а не все числа, разделенные пробелами;

Поэтому вряд ли разумно полагаться на режим совместимости при разработке новых таблиц стилей, ориентированных на процессоры для XSLT 2.0.

6.6. Эмуляция объектно-ориентированных методик повторного использования и паттернов проектирования

Задача

Хотелось бы перейти от «копирования и вставки» к созданию библиотек повторно используемого XSLT-кода.

Решение

Понятно, что XSLT – не объектно-ориентированный язык. Однако объектная ориентация – это, скорее, принципы конструирования обобщенного повторно используемого кода, а не просто классы, объекты, наследование, инкапсуляция и т.д.

В XSLT 2.0 появилось два механизма, упрощающих разработку в объектно-ориентированном стиле. Первый – команда `xsl:next-match`, второй – туннельные параметры. Команда `xsl:next-match` – это обобщение команды `xsl:apply-imports` в XSLT 1.0. Напомним, что последняя используется для вызова шаблонов с более низким уровнем приоритета. Команда `xsl:next-match` обобщает это поведение, позволяя вызывать подходящие низкоприоритетные шаблоны, определенные в той же или импортирующей таблице стилей. Этот механизм сродни вызову метода базового класса в объектно-ориентированном программировании.

```
<xsl:template match="author | title | subtitle | deck" priority="1">
  <a name="{generate-id()}">
    <span class="{name()}">
      <xsl:apply-templates/>
    </span>
  </a>
</xsl:template>

<xsl:template match="author" priority="2">
  <div>
    <span class="by">By </span>
    <xsl:next-match/>
  </div>
</xsl:template>

<xsl:template match="title" priority="2">
```

```
  <h1 class="title"><xsl:next-match/></h1>
</xsl:template>

<xsl:template match="deck" priority="2">
  <h2 class="deck"><xsl:next-match/></h2>
</xsl:template>

<xsl:template match="subtitle" priority="2">
  <h2 class="subtitle"><xsl:next-match/></h2>
</xsl:template>
```

Еще одно нововведение в XSLT 2.0 – возможность передавать параметры командам `xsl:next-match` и `xsl:apply-imports`:

```
<xsl:next-match>
  <xsl:with-param name="indent" select="2"/>
</xsl:next-match>
```

Кроме того, в XSLT 2.0 появились *туннельные параметры* – разновидность динамических областей видимости, столь популярных в функциональном программировании. С помощью этого механизма можно передать команде `xsl:apply-templates` параметры, неизвестные тем шаблонам, которые она сочтет нужным применить. Однако эти параметры будут прозрачно передаваться от одного вызова другому, пока не достигнут шаблона, знающего о них. Отметим, что атрибут `tunnel="yes"` следует указывать как в точке вызова, так и в точке приема параметров:

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Стандартные правила обработки документа -->
<xsl:import href="doc.xslt"/>

<!-- Параметр, который автор doc.xslt не предвидел -->
<xsl:param name="customParam"/>

<xsl:template match="/">
 <!-- Вызываем шаблоны из doc.xslt, которые ничего не знают о
customParam -->
  <xsl:apply-templates>
     <xsl:with-param name="customParam"
                    select="$customParam" tunnel="yes"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="heading1">
```

```
    <!-- Сделать с элементами heading1 что-то нестандартное
    в зависимости от значения customParam -->
     <xsl:param name="customParam" tunnel="yes"/>
     <!-- ...
                  -->
</xsl:template>

</xsl:stylesheet>
```

Это чрезвычайно важное дополнение, поскольку оно позволяет отделить шаблоны, специфичные для конкретного приложения, от общих шаблонов, не вводя глобальных параметров или переменных.

Обсуждение

В методиках объектно-ориентированной разработки очень популярны паттерны проектирования. Это испытанные и доказавшие свою состоятельность приемы, применимые к решению широкого спектра задач. Паттерны упрощают взаимодействие между разработчиками, так как позволяют назвать использованный подход к решению и контекст, в котором они применим, более или менее стандартизованным именем.

Средства, описанные в этом рецепте и в рецепте 6.3, можно использовать для реализации некоторых стандартных паттернов на языке XSLT. Ниже мы показываем, как их можно адаптировать.

Паттерн Template Method (Шаблонный метод)

Определяется скелет таблицы стилей для выполнения операции, а конкретные шаги делегируются шаблонам, которые реализованы в импортирующих таблицах. Паттерн Template Method позволяет другим разработчикам переопределить некоторые шаги, не изменяя общую структуру преобразования.

Рассмотрим таблицу стилей, определяющую стандарт, которому компания следует при публикации XML-документа в Web:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xsl:output method="xhtml" indent="yes"/>

  <xsl:param name="titlePrefix" select=" '' " as="xs:string"/>

  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of
                  select="concat($titlePrefix, /*/title)"/></title>
```

```
      </head>
      <body>
          <xsl:call-template name="standard-processing-sequence"/>
      </body>
    </html>
    </xsl:template>

  <xsl:template name="standard-processing-sequence">
    <xsl:apply-templates mode="front-matter">
      <xsl:with-param name="mode" select=" 'front-matter' "
                      tunnel="yes" as="xs:string"/>
    </xsl:apply-templates>

    <xsl:apply-templates mode="toc">
      <xsl:with-param name="mode" select=" 'toc' "
                      tunnel="yes" as="xs:string"/>
    </xsl:apply-templates>

    <xsl:apply-templates mode="body">
      <xsl:with-param name="mode" select=" 'body' "
                      tunnel="yes" as="xs:string"/>
    </xsl:apply-templates>

    <xsl:apply-templates mode="appendicies">
      <xsl:with-param name="mode" select=" 'appendicies' "
                     tunnel="yes" as="xs:string"/>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="/*" mode="#all">
    <xsl:param name="mode"  tunnel="yes" as="xs:string"/>
    <div class="{$mode}">
      <xsl:apply-templates mode="#current"/>
    </div>
  </xsl:template>

  <!-- Здесь могут быть шаблоны, применяемые по умолчанию в различных
  режимах. Однако в импортирующих таблицах стилей их можно
  переопределить. -->

</xsl:stylesheet>
```

Здесь мы воспользовались режимами для определения каждого этапа обработки. А название текущего режима еще и передается в туннельном параметре.

У такого подхода есть два преимущества. Во-первых, он полезен для отладки шаблонов, которые сопоставляются в нескольких режимах. Во-вторых, если поведение шаблонов с несколькими режимами отличается незначительно, то отличия можно реализовать с помощью функций от режима, ничего не зная о специфике конкретного режима. Например, если шаблон присоединяет CSS-стили к выходным элементам, то в названия стилей можно в качестве префикса добавить название режима или придумать еще какое-то общее отображение режима на стиль (например, в виде справочной таблицы).

Паттерн Chain of Responsibility (Цепочка обязанностей)

Назначение этого паттерна – разорвать связь между инициатором преобразования с шаблонами, которые это преобразование реализуют, предоставив нескольким шаблонам шанс обработать запрос. Для выбора подходящего шаблона используется механизм сопоставления, а не условная логика.

Ключом к переносимости этого паттерна служат приоритеты, поскольку некоторые процессоры XSLT отказываются обрабатывать шаблоны с неоднозначными правилами предшествования. Следующий пример взят из проекта динамического сайта, над которым я работал, применяя систему Cocoon. В этом проекте использовались шаблонные HTML-файлы, а способ преобразования динамического содержимого из базы данных XML в HTML-разметку диктовался атрибутами `class`. Детали шаблонов я опущу, поскольку они не так важны, как общая структура. В данном примере в любом узле `xhtm:td` сопоставляется только один шаблон, но в общем случае можно воспользоваться командой `xsl:next-match` для объединения результатов применения нескольких подходящих шаблонов.

```
<xsl:template match="xhtm:td[matches(@class, '^keep-(\w+)')]"
              mode="template" priority="2.1">
</xsl:template>

<xsl:template match="xhtm:td[matches(@class, '^(flow|list)-(\w+)')]"
               mode="template" priority="2.2">
</xsl:template>

<xsl:template match="xhtm:td[matches(@class, '^repeat-(\w+)')]"
               mode="template" priority="2.3">
</xsl:template>

<xsl:template match="xhtm:td[matches(@class, '^download-(\w+)')]"
               mode="template" priority="2.4">
</xsl:template>

<xsl:template match="xhtm:td[matches(@class, '^keep-(\w+)')]"
              mode="template" priority="2.1">
<xsl:template>
```

Паттерн Decorator (Декоратор)

Добавить поведение в низкоприоритетные шаблоны путем сопоставления узлам шаблонов с более высоким приоритетом импорта. Вызвать базовое поведение с помощью команды `xsl:next-match`.

Этот классический способ применения следующего соответствия мы уже обсуждали в разделе «Решение», поэтому здесь обойдемся без примера.

6.7. Обработка неструктурированного текста с помощью регулярных выражений

Задача

Требуется преобразовать XML-документ, содержащий фрагменты неструктурированного текста, которые должны быть надлежащим образом размечены.

Решение

Для работы с регулярными выражениями в XSLT 2.0 есть три функции: `match()`, `replace()` и `tokenize()`. Мы рассматривали их в главе 1. Появилась также новая команда `xsl:analyze-string()`, которая позволяет обрабатывать текст еще более интересными способами.

У команды `xsl:analyze-string()` имеется атрибут `select` для задания подлежащей обработке строки, атрибут `regex` для задания применяемого к строке регулярного выражения и необязательный атрибут `flags` для более точного указания того, как должно применяться регулярное выражение. Ниже перечислены стандартные флаги:

- `i` – режим учета регистра;
- `m` – многострочный режим, в котором метасимволы ^ и $ сопоставляются с началом и концом линейных строк (lines), а не строки (string) в целом (режим по умолчанию);
- `s` – метасимвол . сопоставляется с символами новой строки (`
`). По умолчанию это не так. Иногда этот режим называют однострочным, но из определения должно быть ясно, что это не противоположность многострочному режиму; флаги `m` и `s` можно задавать совместно;
- `x` – в регулярном выражении допускается в качестве разделителей использовать пробелы, которые перестают быть значащими символами.

Для обработки подстроки, сопоставившейся с регулярным выражением, применяется дочерний элемент `xsl:matching-substring`, а для обработки несопоставившейся строки – элемент `xsl:non-matching-substring`. Любой из них может быть опущен. Можно также ссылаться на запомненные группы (части регулярного выражения, заключенные в круглые скобки), для чего служит функция `regex-group`, вызываемая внутри `xsl:matching-substring`:

```
<xsl:template match="date">
  <xsl:copy>
    <xsl:analyze-string select="normalize-space(.)"
       regex="(\d\d\d\d) ( / | - ) (\d\d) ( / | - ) (\d\d)"
       flags="x">
      <xsl:matching-substring>
        <year><xsl:value-of select="regex-group(1)"/></year>
        <month><xsl:value-of select="regex-group(3)"/></month>
        <day><xsl:value-of select="regex-group(5)"/></day>
      </xsl:matching-substring>
      <xsl:non-matching-substring>
        <error><xsl:value-of select="."/></error>
      </xsl:non-matching-substring>
    </xsl:analyze-string>
  </xsl:copy>
</xsl:template>
```

Полезным дополнением к `xsl:analyze-string()` является функция `unparsed-text()`. Она позволяет считывать содержимое текстового файла как строку. Следовательно, файл не анализируется и потому не обязательно должен быть представлен в формате XML. Вообще, использовать эту функцию для чтения XML-файлов имеет смысл разве что в исключительных случаях.

Следующая таблица стилей преобразовывает простой файл с полями, разделенными запятыми, (без заключенных в кавычки строк) в формат XML:

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/02/xpath-functions"
xmlns:xdt="http://www.w3.org/2005/02/xpath-datatypes">

 <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

 <xsl:param name="csv-file" select=" 'test.csv' "/>

  <xsl:template match="/">

    <converted-csv filename="{$csv-file}">
      <xsl:for-each select="tokenize(unparsed-text($csv-file, 'UTF-8'),
                                     '\n')">
        <xsl:if test="normalize-space(.)">
          <row>
              <xsl:analyze-string select="." regex="," flags="x">
                <xsl:non-matching-substring>
                  <col><xsl:value-of select="normalize-space(.)"/></col>
```

```
            </xsl:non-matching-substring>
          </xsl:analyze-string>
        </row>
      </xsl:if>
    </xsl:for-each>
  </converted-csv>

</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Добавленный в XSLT 2.0 механизм регулярных выражений вкупе с функцией `unparsed-text()` открывает целый ряд новых возможностей, которые были практически недоступны в XSLT 1.0. Тем не менее, я стал бы использовать XSLT для обработки документов, представленных не в формате XML, только в том случае, когда мультиязычное решение (например, Java и XSLT или Perl и XSLT) по какой-то причине неприемлемо. Разумеется, если XSLT – единственный язык, которым вы хотите владеть в совершенстве, то новые средства – непаханое поле для экспериментов.

Отчасти мое стремление сойти с корабля XSLT при входе в область обработки неструктурированного текста связано с некоторыми чертами, которых недостает команде `xsl:analyze-string`. Хотелось бы, чтобы функции `position()` и `last()` работали внутри элемента `xsl:matching-string` и сообщали, что данное совпадение имеет номер `position()` из `last()` найденных. Иногда я использую цикл `xsl:for-each` по результатам, возвращенным `tokenize()`, вместо `xsl:analyze-string`, но и это решение не идеально, поскольку возвращаются только не сопоставившиеся части. Но часто для сложного анализа, где число возможных сопоставлений с регулярным выражением, содержащим альтернативы (`|`), велико, не существует никакого другого выхода, кроме как воспользоваться `xsl:analyze-string`. Однако невозможно сказать, с каким регулярным выражением произошло сопоставление без повторного опроса с помощью функции `match()`, а на мой вкус это избыточно и расточительно, поскольку сам-то механизм regex наверняка знает, с какой частью строки только что произошло сопоставление:

```
<xsl:template match="text()">
  <xsl:analyze-string select="."
      regex=' [\-+]?\d\.\d+\s*[eE][\-+]?\d+                    |
              [\-+]?\d+\.\d+                                   |
              [\-+]?\d+                                        |
              "[^"]*?"                                         '
      flags="x">
    <xsl:matching-substring>
```

```
        <xsl:choose>
           <xsl:when test="matches(.,'[\-+]?\d\.\d+\s*[eE][\-+]?\d+')">
            <scientific><xsl:value-of select="."/></scientific>
            </xsl:when>
           <xsl:when test="matches(.,'[\-+]?\d+\.\d+')">
            <decimal><xsl:value-of select="."/> </decimal>
          </xsl:when>
          <xsl:when test="matches(.,'[\-+]?\d+')">
            <integer><xsl:value-of select="."/> </integer>
          </xsl:when>
          <xsl:when test='matches(.," "" [^""]*? "" ", "x")'>
            <string><xsl:value-of select="."/></string>
          </xsl:when>
        </xsl:choose>
    </xsl:matching-substring>
  </xsl:analyze-string>
</xsl:template>
```

Впрочем, критиковать всякий горазд. Разумеется, когда совершенствуешь язык, приходится преодолевать самые разнообразные проблемы и идти на компромиссы. Тем не менее, при всем уважении к комитету, работавшему над спецификацией XSLT 2.0, я хотел бы, чтобы команда `xsl:analyze-string` работала следующим образом:

```
<!-- ЭТО НЕ КОРРЕКТНЫЙ XSLT 2.0, а лишь мечты авторы -->
<xsl:template match="text()">
  <xsl:analyze-string select="."
                      flags="x">
    <xsl:matching-substring regex="[\-+]?\d\.\d+\s*[eE][\-+]?\d+">
      <scientific><xsl:value-of select="."/></scientific>
    </xsl:matching-substring>
    <xsl:matching-substring regex="[\-+]?\d+\.\d+'">
      <decimal><xsl:value-of select="."/> </decimal>
    </xsl:matching-substring>
    <xsl:matching-substring regex=" [\-+]?\d+')">
      <integer><xsl:value-of select="."/> </integer>
    </xsl:matching-substring>
    <xsl:matching-substring regex=' "[^"]*?" '>
      <string><xsl:value-of select="."/></string>
    </xsl:matching-substring>
    <xsl:non=matching-substring>
      <other><xsl:value-of select="."/></other>
   </xsl:non=matching-substring>
  </xsl:analyze-string>
</xsl:template>
```

6.8. Решение трудных задач сериализации с помощью таблиц символов

Задача

Необходим точный контроль над сериализацией документа, а имеющаяся в XSLT 1.0 команда `disable-output-escaping` предоставляет слишком ограниченные возможности.

Решение

В XSLT 2.0 появились так называемые таблицы символов, обеспечивающие точный контроль над сериализацией. Эти таблицы предназначены для использования вместе с командой `xsl:output`.

Команда `xsl:character-map` обладает следующими атрибутами:

`name`

Задает имя таблицы символов.

`use-character-maps`

Список других таблиц символов, включаемых в данную.

Содержимым элемента `xsl:character-map` является последовательность элементов `xsl:output-character`. Они определяют отображение между одним символов Unicode и строкой, выводимой взамен него при сериализации этого символа. Следующую таблицу можно использовать для вывода различных специальных символов пропуска в виде компонентов:

```
<xsl:character-map name="spaces">
    <xsl:output-character char=" " string="&amp;npsp;"/>
    <xsl:output-character char=" " string="&amp;emsp;"/>
    <xsl:output-character char=" " string="&amp;numsp;"/>
    <xsl:output-character char=" " string="&amp;puncsp;"/>
    <xsl:output-character char=" " string="&amp;thincsp;"/>
    <xsl:output-character char=" " string="&amp;hairsp;"/>
</xsl:character-map>
```

Есть еще одно более тонкое применение таблицы символов – вывод нестандартных документов, которые иначе было бы трудно создать, потому что они нарушают правила XML или XSLT. В третьем издании своей книги *XSLT Programmer's Reference* Майкл Кэй приводит пример вывода закомментированных элементов. Пример ниже – это вариация на ту же тему. Идея заключается в том, чтобы сгенерировать копию входного документа, закомментировав некоторые элементы согласно синтаксису XML:

```
<?xml version="1.0"?>
<!-- Определяем для себя компоненты, пользуясь символами из частного
диапазона Unicode -->
```

```
<!DOCTYPE xsl:stylesheet [
  <!ENTITY start-comment "&#xE501;">
  <!ENTITY end-comment "&#xE502;">
]>

<xsl:stylesheet version="2.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Импортируем тождественное преобразование -->
<xsl:import href="copy.xslt"/>

<!-- Просим сериализатор применять нашу таблицу символов, которая
определена ниже -->
<xsl:output use-character-maps="comment-delimiters"/>

<!-- Определяем ключ, который будет использоваться для идентификации
элементов, подлежащих комментированию -->
<xsl:key name="omit-key" match="omit" use="@id"/>

<!-- Отображаем определенные выше компоненты на строки, образующие
начало и конец комментария в синтаксисе XML -->
<xsl:character-map name="comment-delimiters">
  <xsl:output-character character="&start-comment;" string="&lt;!--"/>
  <xsl:output-character character="&end-comment;" string="--&gt;"/>
</xsl:character-map>

<!-- Комментируем те элементы, для которых атрибут id соответствует id
элемента omit из внешнего документа omit.xml -->
<xsl:template match="*[key('omit-key',@id,doc('omit.xml'))]">
  <xsl:text>&start-comment;</xsl:text>
  <xsl:copy>
    <xsl:apply select="@* | *"/>
  </xsl:copy>
  <xsl:text>&end-comment;</xsl:text>
</xsl:template>
```

См. также

Эван Ленц разработал конвертер из XML в строку, который позволяет по-другому решить задачу точного контроля над сериализацией. См. *http://xmlportfolio.com/xml-to-string/*.

6.9. Вывод нескольких документов

Задача

Необходима переносимая таблица стилей для вывода нескольких документов.

Решение

Хотя в большинстве реализаций XSLT 1.0 имеются расширения, помогающие обрабатывать несколько документов, все они немного различаются между собой. А в XSLT 2.0 есть команда `xsl:result-document`.

Эта команда обладает следующими атрибутами:

`format`

Задает имя формата вывода, объявленного в именованной команде `xsl:output`.

`href`

Определяет место назначения выходного документа.

`validation`

Определяет способ контроля результирующего дерева.

`type`

Определяет тип, используемый для контроля результирующего дерева.

Приведенный ниже код разбивает XML-документ на несколько документов в соответствии с группами, которые формирует команда `xsl:for-each-group`. Каждому выходному документу присваивается имя, в которое в качестве суффикса добавлен ключ группировки:

```
<xsl:template match="products">
  <xsl:for-each-group select="product" group-by="@type">
    <xsl:result-document href="prod-{current-grouping-key()}.xml">
    <xsl:copy-of select="current-group()"/>
    </xsl:result-document>
  </xsl:for-each-group>
</xsl:template>
```

Иногда возникает необходимость в таблице стилей, которая выводит документы в разных форматах. Например, по умолчанию принимается формат XML, но допускается также вывод в виде HTML. Для этого воспользуемся имеющимся в XSLT 2.0 механизмом для задания нескольких форматов вывода:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- По умолчанию принимается формат XML -->
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- Другой именованный формат для вывода в виде HTML -->
  <xsl:output method="html" encoding="UTF-8" indent="yes" name="html-out"/>
```

```
  <xsl:template match="/">
    <xsl:apply-templates/>
    <xsl:result-document href="result.html" format="html-out">
       <xsl:apply-templates mode="html"/>
    </xsl:result-document>
  </xsl:template>

  <!-- Прочие шаблоны -->

</xsl:stylesheet>
```

Обсуждение

Хотя в самой команде <xsl:result-document> нет ничего сложного, потенциально ее можно спутать с другой командой XSLT 2.0 – <xsl:document>. Связано это с тем, что в спецификации XSLT 1.1 (ныне объявленной недействующей) также определена команда <xsl:document>, функционирующая подобно <xsl:result-document> из версии 2.0.

В версии 2.0 команда <xsl:document> играет более скромную роль. Ее назначение – сконструировать документный узел, например для того, чтобы выполнить контроль на уровне документа, не прибегая к сериализации. Обычно результат работы <xsl:document> запоминается в переменной:

```
<xsl:variable name="tempDoc" as="document(element(*, my:document))">
    <xsl:document type="my:document" validation="strict">
         <xsl:apply-templates select="/*"/>
    </xsl:document>
</xsl:variable>
```

Если позже вы захотите вывести этот документ, то сможете сделать это с помощью команды <xsl:result-document>:

```
<xsl:result-document href="doc.xml">
    <xsl:copy-of select="$tempDoc"/>
</xsl:result-document>
```

См. также

О том, как вывод нескольких документов реализуется в XSLT 1.0 с помощью расширений, см. рецепт 8.6.

6.10. Обработка строковых литералов, содержащих кавычки

Задача

В XSLT 1.0 обрабатывать строковые литералы, содержащие кавычки, трудно, поскольку не существует никакого символа экранирования.

Решение

Эта проблема решается за счет нового соглашения: чтобы экранировать символ кавычки, нужно ввести его два раза подряд. Ниже мы пытаемся выполнить сопоставление со строками, заключенными в двойные или одинарные кавычки. Атрибут `test` мы заключили в одинарные кавычки, следовательно, строковый литерал для регулярного выражения нужно заключить в двойные. Поэтому все двойные кавычки внутри этого литерала необходимо продублировать. Правила XML вынуждают нас использовать компонент `'` вместо ', но это лишь подчеркивает тот факт, что экранирование средствами XML – отдельный вопрос, само по себе это не помогает решить задачу. Другими словами, если заменить "" на `"`, то анализатор XML будет доволен, но анализатору XSLT этого мало.

```
<xsl:if test=" matches(., ' '' [^''] '' | &apos;[^&apos;] &apos;  ','x') ">
</xsl:if>
```

Эквивалентное решение выглядит так:

```
<xsl:if test=" matches(., ' &quot; [^&quot;] &quot; | ''[^''] ''  ','x') ">
</xsl:if>
```

Обсуждение

Отсутствие символа экранирования в XSLT 1.0 было большим неудобством, но его можно было обойти с помощью переменных и конкатенации:

```
<xsl:variable name="d-quote" select="'"'/>
<xsl:variable name="s-quote" select="'"'/>
<xsl:value-of select="concat('He said,', $d-quote, 'John', $s-quote, 's',
'dog turned green.', $d-quote)"/>
```

6.11. Новые возможности старых конструкций XSLT 1.0

Задача

В XSLT 2.0 появилось немало мелких усовершенствований, и отыскать их все сразу довольно трудно.

Решение

Многие средства XSLT 2.0 стали развитием уже имевшихся в XSLT 1.0 команд и функций. И уловить различия не так просто, как в случае совсем новых средств. В этом разделе мы перечислим все такие нюансы.

xsl:apply-templates

- Атрибут `mode` может принимать значение `#current`, которое означает, что обработка должна продолжаться в текущем режиме.

- Можно воспользоваться поддержкой последовательностей, если написать список значений, разделенных запятыми (например, `xsl:apply-templates select="title, heading, para"/>`, когда нужно обработать сначала элементы `title`, потом `heading` и в самом конце `para`. В XSLT 1.0 пришлось бы написать три отдельных команды `apply-templates`.

В обеих версиях можно написать:

```
<xsl:apply-templates select "title | heading | para"/>
```

но при этом не задается порядок обработки. Дочерние узлы будут обрабатываться в том порядке, в котором находятся в документе, а не в порядке перечисления их в атрибуте `select`.

xsl:attribute

- Головной болью многих программистов на XSLT 1.0 была невозможность написать такую конструкцию: `<xsl:attribute name="foo" select="10"/>`, поскольку атрибут `select` в этом контексте не поддерживается. Теперь это исправлено, поэтому для задания простых атрибутов следует предпочесть такой способ, хотя синтаксис конструктора последовательности тоже сохранился.
- Для процессоров, поддерживающих схемы, добавлены атрибуты `type` и `validation`. Атрибут `type` задает встроенные или определенные пользователем типы, описываемые схемой W3C. `Validation` говорит, как следует проверять атрибут.

xsl:call-template

- Результат вызова шаблона может быть произвольной последовательностью (например, целых чисел), а не только набором узлов.
- Если передать (с помощью `xsl:with-param`) параметр, который не определен в вызываемом шаблоне, возникнет ошибка.

xsl:comment

- Как и в случае `xsl:attribute`, в дополнение к конструктору последовательности поддерживается атрибут `select`.

xsl:copy и xsl:copy-of

- Определен новый атрибут `copy-namespace="yes" | "no"`, позволяющий указать, следует ли копировать узлы пространств имен, принадлежащие копируемому элементу. По умолчанию подразумевается значение «yes», совместимое с поведением в версии 1.0.
- Добавлен атрибут `type` для задания встроенных или определенных пользователем типов, описываемых схемой W3C.
- Добавлен атрибут `validation`, описывающий, как следует проверять результат и надо ли сохранять существующие аннотации типа.

xsl:element

- Для процессоров, поддерживающих схемы, добавлены атрибуты `type` и `validation`. Атрибут `type` задает встроенные или определенные пользователем типы, описываемые схемой W3C. `Validation` говорит, как следует проверять атрибут.

xsl:for-each

- Может обрабатывать произвольные последовательности, а не только последовательности узлов:

```
<xsl:for-each select="(1, 2, 3, 4, 5)">
    <xsl:value-of select="."/><xsl:text>)&#xa;</xsl:text>
</xsl:for-each>
```

xsl:key

- Атрибуты `match` и `use` теперь могут ссылаться на глобальные переменные при условии, что не существует циклических зависимостей между значениями переменных и ключом:

```
<xsl:variable name="state" select=" 'active' "/>
<xsl:key name="state-key" match="employee[@state=$state]" use="@type"/>
```

- Вместо атрибута `use` можно использовать значение, формируемое конструктором последовательности:

```
<!-- определение значения ключа путем нетривиальной обработки -->
<xsl:key name="sick-key" select="employee">
    <xsl:apply-templates select="record[@type='sick-day']" mode="sick-
key"/>
</xsl:key>
```

- Для определения того, когда два значения ключа совпадают, можно указать таблицу сравнения. Состав таблиц сравнения зависит от реализации.

xsl:message

- Значением атрибута `terminate` теперь может быть выражение. Это существенно упрощает жизнь, когда нужно глобально изменить поведение программы в случае завершения.
- Помимо конструктора последовательности, поддерживается также атрибут `select`:

```
<xsl:param name="terminate" select=" 'no' "/>

<xsl:template match="employee">
  <xsl:if test="not(@type)">
     <xsl:message terminate="{$terminate}"
           select=" Отсутствует атрибут type для элемента employee' "/>
```

```
    <xsl:if>
</xsl:template>
```

xsl:number

- Добавлен атрибут `select`, чтобы можно было нумеровать и другие узлы, помимо контекстного.
- Параметры форматирования расширены, так что теперь можно выводить слова типа «one», «two», «three» (на выбранном языке):

```
<-- Выводится 'Ten' -->
<xsl:number value="10" format="Ww"/>

<-- Выводится 'ten' -->
<xsl:number value="10" format="w"/>

<- Выводится 'TEN' ->
<xsl:number value="10" format="W"/>
```

xsl:output

- Можно назначить имя, которое будет использоваться в команде `xsl:result-document`. Детали см. в рецепте 6.10.
- Поддерживается новый метод вывода XHTML.
- Новый атрибут `escape-uri-attributes` определяет, нужно ли кодировать атрибуты URI при выводе в формате HTML или XHTML.
- Новый атрибут `include-content-type` определяет, нужно ли добавлять в выходной документ элемент `<meta>`, описывающий тип и кодировку содержимого.
- Новый атрибут `normalize-unicode` определяет, нужно ли нормализовать символы Unicode. Дополнительную информацию о нормализации см. в документе *http://www.w3.org/TR/2002/WD-charmod-20020430/*.
- Новый атрибут `undeclare-namespaces` определяет, следует ли отменять объявления пространств имен в XML 1.1, когда они выходят из области видимости. Объявление пространства имен отменяется директивой `xmlns:`*`pre`*`=""`, где *pre* – некоторый префикс.
- Новый атрибут `use-character-maps` позволяет задать список имен таблиц символов. См. рецепт 6.8.

xsl:param

- Для задания типа параметра можно использовать атрибут `as`.
- Атрибут `required` позволяет указать, является параметр обязательным или нет.
- Атрибут `tunnel` означает, что для данного параметра поддерживается туннелирование. См. рецепт 6.6.

xsl:processing-instruction

- Поддерживается атрибут `select`.

xsl:strip-space

- Атрибут `elements` теперь поддерживает задание имен в виде *`*:Name`*, означающем, что пробелы следует исключать из всех элементов с именем *`Name`* вне зависимости от пространства имен. См. рецепт 7.1.

xsl:stylesheet

- Новый атрибут `default-validation` определяет способ контроля по умолчанию, применяемый при создании новых узлов элементов и атрибутов, когда в команде, создающей эти узлы, отсутствует атрибут `validation`.
- Новый атрибут `xpath-default-namespace` определяет пространство имен, подразумеваемое для элементов без префиксов в выражениях XPath.

xsl:template

- Для задания нескольких режимов в атрибуте `mode` можно указать значение `#all` или список режимов.
- Атрибут `as` позволяет указать тип результата.
- Атрибут `match` поддерживает сопоставление по типу.
- Атрибут `match` может ссылаться на глобальные переменные или параметры.

xsl:value-of

- Новый атрибут `separator` позволяет разделять последовательности. См. рецепт 7.2.
- Помимо атрибута `select`, поддерживается также синтаксис конструктора последовательности.

xsl:variable

- Для задания типа переменной можно использовать атрибут `as`.

xsl:with-param

- Для задания типа параметра можно использовать атрибут `as`.
- Атрибут `tunnel` указывает, что для данного параметра поддерживается туннелирование. См. рецепт 6.6.
- Теперь можно использовать совместно с командой `xsl:apply-imports` и с новой командой `xsl:next-match`.

current()

- Эта функция может возвращать элементы общего вида (например, строки), а не только узлы.
- Может использоваться совместно с образцом для ссылки на сопоставившийся элемент.

```
<!-- Сопоставляется с элементами-потомками, у которых есть атрибуты, чьи
значения соответствуют локальному имени рассматриваемого в данный момент
элемента -->
<xsl:template match="*[descendant::*/@* = local-name(current())]">
```

document()

- Первый аргумент может быть произвольной последовательностью URI.

Новая функция XPath `doc()` представляет собой упрощенную альтернативу функции XSLT `document()`.

function-available()

- Принимает еще и второй аргумент, который задает *арность* (число аргументов) проверяемой функции.

key()

- Необязательный третий аргумент служит для задания документа, в котором нужно производить поиск.

Это устраняет необходимость в командах `xsl:for-each`, единственное назначение которых – переключиться на новый документ, чтобы функцию `key()` можно было использовать относительно этого документа:

```
<!-- Такой код больше не нужен -->
<xsl:for-each select="doc('other.xml')">
    <xsl:if test="key('some-key', $val)">
            <!-- ...-->
    </xsl:if>
</xsl:for-each>

<!-- Пишите так -->
<xsl:if test="key('some-key', $val,
                doc('other.xml'))">
    <!-- ...-->
</xsl:if>
```

system-property()

- Определено свойство `xsl:product`, которое возвращает название процессора XSLT (например, Saxon).
- Определено свойство `xsl:product-version`, которое возвращает версию процессора XSLT (например, 8.1).
- Определено свойство `xsl:is-schema-aware`, которое возвращает `yes` или `no` в зависимости от того, поддерживает процессор схемы или нет.
- Определено свойство `xsl:supports-serialization`, которое возвращает `yes` или `no` в зависимости от того, поддерживает процессор сериализацию или нет.
- Определено свойство `xsl:supports-backward-compatibility`, которое возвращает `yes` или `no` в зависимости от того, поддерживает процессор режим обратной совместимости или нет (например, атрибут version="1.0").

Обсуждение

Основным мотивом для совершенствования старых команд XSLT было добавление поддержки типов и согласованности. Под согласованностью понимается главным образом добавление атрибута `select` в тех местах, где раньше поддерживался только конструктор последовательности, и наоборот.

Одно из странных решений, принятых создателями XSLT, состоит в том, что они продолжают использовать два разных названия там, где, на мой взгляд, хватило бы и одного. В частности, речь идет об атрибутах `as` и `type`. Они никогда не используются совместно, так почему бы не оставить только `type`? В контексте XSLT 1.0 такой же аргумент можно было бы привести в пользу отказа от конструкции `xsl:with-param` и оставления лишь `xsl:param`.

Глава 7. Преобразование XML в текст

Средства обработки текста позволяют красиво оформить любую мысль, даже такую, которую оформлять вообще не стоило бы.

Дж. Финеган

7.0. Введение

Понятно, что в век Интернета результатом преобразований с помощью XSL и XSLT являются преимущественно такие форматы, как HTML, XHTML, XML и PDF. Но и старый добрый текст никогда не устареет, поскольку это общий знаменатель, понятный и компьютеру, и человеку. XML-документы часто преобразуют в текстовую форму для импорта в другие программы, которые вовсе не умеют интерпретировать XML или делают это не так, как вам нужно. Текст выводится и в тех случаях, когда результат нужно отправить на терминал или подвергнуть последующей обработке, например, через конвейер UNIX.

Многие примеры в этой главе – иллюстрация приемов создания обобщенных конвертеров XML в текст. Слово «обобщенные» подразумевает, что преобразование можно легко настроить на чтение различных входных XML-данных, на вывод в разных форматах или на то и другое одновременно. Рассматриваемая техника находит применение и за пределами конкретного рецепта, а часто даже за пределами сферы обработки текста. В частности, рекомендую познакомиться с рецептами 7.2 – 7.5, даже если в данный момент перед вами не стоит похожая задача.

Из всех форматов вывода, поддерживаемых командой `xsl:output`, текст наиболее чувствителен к пробелам. Поэтому теме пустого пространства посвящен отдельный рецепт 7.1. Программисты, не очень хорошо знакомые с XML и XSLT, часто возмущаются тем, как непоследовательно трактуется пустое пространство. Но, разобравшись с правилами и способами их применения, вы увидите, что отформатировать вывод нужным образом не так уж сложно.

Одной из областей, где имеет смысл применять преобразование XML в текст, является, на мой взгляд, генерация исходного кода. Но эта тема затрагивает вопросы, выходящие за пределы простого преобразования и форматирования. Поэтому мы рассмотрим ее отдельно в главе 10.

7.1. Пустое пространство

Задача

Требуется преобразовать XML в форматированный текст, но из-за странностей в обработке символов пропуска все попытки форматирования терпят неудачу.

Решение

Рассмотрим следующий аннотированный пример XML-документа. Символы ↵ (новая строка), → (табуляция) и ? (пробел) обозначают текстовые узлы, содержащие только пустое пространство, на которые часто не обращают внимания, хотя они тоже могут копироваться в выходной документ:

```
<review>↵
→<author>↵
→→ <name>Sal Mangano</name>↵
→→<email>smangano@somewhere.com</email>↵
→</author>↵
<title>XSLT Cookbook</title>↵
<reviewer>↵
→→<name>?<annonymous/>?</name>↵
→→<email>smangano@somewhere.com</email>↵
→→<comment>↵
Totally awesome. <b>Worth every cent!</b>?<i>Must buy because I ↵
know the author peronally and he can sure use the money.</i>↵
→→</comment>↵
→</reviewer>↵
</review>↵
```

Слишком много пустого пространства

1. Воспользуйтесь командой `xsl:strip-space`, чтобы избавиться от узлов, содержащих только символы пропуска.

   Этот элемент верхнего уровня имеет единственный атрибут `elements`, значением которого является список разделенных пробелами имен элементов, в которых вы хотели бы убрать лишнее пустое пространство. Под «лишним пустым пространством» понимаются текстовые узлы, содержащие только символы пропуска. Это означает, что пустое пространство, разделяющее слова в элементе `comment` выше, не лишнее, так как в нем есть не только символы пропуска. С другой стороны, символы пропуска, обозначенные специальными символами, – лишнее пустое пространство.

   Широко распространена следующая идиома: в начале задается команда `<xsl:strip-space elements="*"/>`, чтобы по умолчанию убрать пустое пространство из всех элементов, а затем с помощью команды `xsl:preserve-space` перечисляют элементы, для которых этого делать не надо. В XSLT 2.0 разрешено также задавать атрибут `elements` в виде `"*:Name"`. Это означает, что процессор должен убрать пустое пространство из всех элементов с данным локальным именем независимо от пространства имен.
2. Воспользуйтесь функцией `normalize-space`, чтобы избавиться от лишнего пустого пространства.

   Типичная ошибка – предполагать, что команда `xsl:strip-space` позаботится обо всем «лишнем» пустом пространстве, включая и то, что

использовалось для выравнивания текста в элементе comment. Это совсем не так. Анализатор всегда считает, что в тексте, где символы пропуска встречаются наряду с другими, всякое пустое пространство значимо. Чтобы удалить лишнее, применяйте функцию normalize-space, например: <xsl:value-of select="normalize-space(comment)"/>.

3. Воспользуйтесь для удаления всех символов пропуска функцией translate.

   Еще одна распространенная ошибка – предполагать, что функция normalize-space удаляет все символы пропуска. Вовсе нет, она лишь убирает такие символы в начале и в конце, а вместо подряд идущих символов пропуска оставляет один пробел. Если нужно удалить все вообще символы пропуска, применяйте функцию translate, например: translate(something,'

 	', "").

4. Применяйте элемент xsl:text, если нужно, чтобы завершающие символы пропуска в таблице стилей не считались значимыми.

   Обычно элемент xsl:text считается способом сохранить пустое пространство. Но если разместить такие элементы в стратегически важных точках, то они могут воспрепятствовать интерпретации завершающих символов пропуска как значимых.

Сравним результаты применения таблицы стилей из примера 7.2 к документу из примера 7.1 в двух режимах:

Пример 7.1. Входной документ

```
<numbers>
    <number>10</number>
    <number>3.5</number>
    <number>4.44</number>
    <number>77.7777</number>
</numbers>
```

Пример 7.2. Обработка чисел с применением и без применения элемента xsl:text

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

<xsl:output method="text"/>
<xsl:strip-space elements="*"/>

<xsl:template match="numbers">
Без пустого текстового элемента:
<xsl:apply-templates mode="without"/>
С пустым текстовым элементом:
<xsl:apply-templates mode="with"/>
```

```
</xsl:template>

<xsl:template match="number" mode="without">
    <xsl:value-of select="."/>,
</xsl:template>

<xsl:template match="number" mode="with">
    <xsl:value-of select="."/>,<xsl:text/>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.3. Результат

```
Без пустого текстового элемента:
10,
3.5,
4.44,
77.7777,

С пустым текстовым элементом:
10,3.5,4.44,77.7777,
```

Отметим, что ничего сверхъестественного в таком применении `xsl:text` нет. Такой же результат получится, если заменить `<xsl:text/>` на `<xsl:if test="0"/>` (но не поступайте так, если только вам не доставляет удовольствия ставить других в тупик). Смысл в том, чтобы поместить узел элемента между запятой и завершающим символом новой строки. Это порождает узел, содержащий только пустое пространство, который будет проигнорирован. Некоторым, впрочем, и такая запись кажется неочевидной, поэтому можете написать `<xsl:text>,</xsl:text>`, если вам так больше нравится.

Слишком мало пустого пространства

1. Применяйте команду `xsl:preserve-space`, чтобы отменить действие `xsl:strip-space` для некоторых элементов.
   Не имеет смысла употреблять команду `xsl:preserve-space` без `xsl:strip-space`. Ведь по умолчанию все символы пропуска во входном документе и в документах, загруженных с помощью функции `document()`, и так сохраняются

Имейте в виду, что процессор MSXML по умолчанию удаляет текстовые узлы, не содержащие ничего, кроме символов пропуска. В этом случае можно воспользоваться командой `xsl:preserve-space`, чтобы исправить указанное расхождение со спецификацией.

2. Используйте команду `xsl:text` для точного задания промежутков в выходном тексте.

Все пустое пространство внутри элемента `xsl:text` сохраняется. Это дает возможность точно контролировать, где должны находиться пробелы. Иногда команда `xsl:text` употребляется для вставки разрывов строк:

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

<xsl:output method="text"/>
<xsl:strip-space elements="*"/>

<xsl:template match="number">
    <xsl:value-of select="."/>
    <xsl:text>&#xa;</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

На некоторых платформах (например, Microsoft) разрыв строки обозначается двумя символами: возврат каретки и перевод строки. Но, поскольку анализатор XML обязан преобразовывать пару таких символов в один символ новой строки, то не существует способа создать платформенно-независимую таблицу стилей. К счастью, в большинстве редакторов для Windows и в утилите командной строки одиночные символы новой строки обрабатываются правильно. Единственное исключение составляет редактор Notepad, входящий в состав дистрибутива Windows.

3. Употребляйте символы неразрываемого пробела.
   В XSLT символ #xA0; (неразрываемый пробел) трактуется не так, как обычные символы пропуска. В частности, и команда xsl:strip-space, и функция normalize-space() его игнорируют. Если в большинстве случаев пустое пространство вам не нужно, но имеются немногочисленные исключения, то можно использовать этот символ во входном документе. Особенно полезен неразрываемый пробел для форматирования HTML-документов, в других контекстах он применяется редко (зависит от программы, которая отображает результат).

Обсуждение

В разделе «Решение» описаны различные приемы управления пустым пространством. Однако полезно также знать формальные правила XSLT, лежащие в основе этих приемов.

Самые важные правила относятся и к таблице стилей, и к документам:

1. Текстовый узел не удаляется, если содержит что-либо, кроме символов пропуска (#x20, #x9, #xD или #xA).
   Хотя атрибут `xml:space` встречается в таблицах стилей и входных документах не так уж часто, нужно понимать, для чего он предназначен.
2. Если для какого-нибудь элемента-предка текстового узла задан атрибут `xml:space` со значением `preserve`, и не существует более близкого предка, у которого атрибут `xml:space` равен `default`, то этот текстовый узел не удаляется, даже если содержит только пустое пространство.

Теперь мы рассмотрим правила, применяемые отдельно к таблицам стилей и к документам. Что касается таблиц стилей, то вариантов не так уж много.

3. Единственный элемент таблицы стилей, для которого узлы, содержащие только пустое пространство, по умолчанию сохраняются, – это `xsl:text`. Здесь «по умолчанию» означает, что противное не задано с помощью команды `xml:space="preserve"`, как описано в п. 2 выше. См. примеры 7.4 и 7.5.

Пример 7.4. Таблица стилей для демонстрации эффекта xsl:text и xml:space="preserve"

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

<xsl:output method="text"/>

<xsl:strip-space elements="*"/>

<xsl:template match="numbers">
Без xml:space="preserve":
<xsl:apply-templates mode="without-preserve"/>
С xml:space="preserve":
<xsl:apply-templates mode="with-preserve"/>
</xsl:template>

<xsl:template match="number" mode="without-preserve">
    <xsl:value-of select="."/><xsl:text> </xsl:text>
</xsl:template>

<xsl:template match="number" mode="with-preserve" xml:space="preserve">
    <xsl:value-of select="."/><xsl:text> </xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.5. Результат

```
Без xml:space="preserve":
10 3.5 4.44 77.7777
С xml:space="preserve":

  10

  3.5

  4.44

  77.7777
```

В первом случае единственное пустое пространство в выходном документе – пробелы внутри элемента `xsl:text`. Но во втором шаблоне задан атрибут `xml:space="preserve"`, и потому сохранены все символы пропуска, включая и оба разрыва строк (первый после `<xsl:template ...>`, второй после `</xsl:text>`).

К входным документам применяются следующие правила:

- Первоначально в список элементов, для которых сохраняется пустое пространство, входят все вообще элементы документа.
- Если имя элемента удовлетворяет условию `NameTest`, заданному в команде `xsl:strip-space`, то он удаляется из списка.
- Если имя элемента удовлетворяет условию `NameTest`, заданному в команде `xsl:preserve-space`, то он добавляется в список.

Условие `NameTest` – это либо простое имя (например, `doc`), либо имя с префиксом пространства имен (например, `my:doc`), либо метасимвол (например, `*`), либо метасимвол с префиксом пространства имен (например, `my:*`). В XSLT 2.0 разрешена также конструкция `*:doc`. Если между командами `xml:strip-space` и `xml:preserve-space` возникают конфликты, то для их разрешения применяются стандартные правила учета приоритета и предшествования при импорте:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:my="http://www.ora.com/XSLTCookbook/ns/my">

<!-- Удалить пустое пространство из всех элементов -->
<xsl:strip-space="*"/>

<!-- кроме входящих в пространство имен "my" -->
<xsl:preserve-space="my:*"/>

<!-- и имеющих имя foo -->
<xsl:preserve-space="foo"/>
```

См. также

Одно из самых полных обсуждений пустого пространства вы найдете в книге Майкла Кэя *XSLT Programmer's Reference* (Wrox, 2001). Там же подробно рассматриваются правила предшествования при импорте.

7.2. Экспорт XML в файл с разделителями полей

Задача

Требуется преобразовать XML-документ в файл, пригодный для импорта в другую программу, например, электронную таблицу.

Решение

Многие приложения умеют импортировать данные с разделителями полей. Самый распространенный формат такого рода называется CSV (Comma Separated Values – значения, разделенные запятыми). Большинство электронных таблиц и баз данных могут обрабатывать этот формат и подобные ему. Отобразить XML на файл с разделителями полей может быть и очень просто, и довольно сложно; все зависит от структуры исходного документа. В этом разделе мы начнем с простых случаев и постепенно перейдем к более сложным.

Создание CSV-файла из плоского документа, в котором данные закодированы в атрибутах

В этом случае элементы отображаются на строки, а атрибуты – на колонки. Эта задача тривиальна. Таблица стилей, показанная в примере 7.7, преобразует документ *people.xml* из примера 7.6 в файл, представленный в примере 7.8.

Пример 7.6. Файл people.xml

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

<people>
  <person name="Al Zehtooney" age="33" sex="m" smoker="no"/>
  <person name="Brad York" age="38" sex="m" smoker="yes"/>
  <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
  <person name="David Willimas" age="33" sex="m" smoker="no"/>
  <person name="Edward Ulster" age="33" sex="m" smoker="yes"/>
  <person name="Frank Townsend" age="35" sex="m" smoker="no"/>
  <person name="Greg Sutter" age="40" sex="m" smoker="no"/>
  <person name="Harry Rogers" age="37" sex="m" smoker="no"/>
  <person name="John Quincy" age="43" sex="m" smoker="yes"/>
  <person name="Kent Peterson" age="31" sex="m" smoker="no"/>
  <person name="Larry Newell" age="23" sex="m" smoker="no"/>
  <person name="Max Milton" age="22" sex="m" smoker="no"/>
  <person name="Norman Lamagna" age="30" sex="m" smoker="no"/>
  <person name="Ollie Kensington" age="44" sex="m" smoker="no"/>
  <person name="John Frank" age="24" sex="m" smoker="no"/>
  <person name="Mary Williams" age="33" sex="f" smoker="no"/>
  <person name="Jane Frank" age="38" sex="f" smoker="yes"/>
  <person name="Jo Peterson" age="32" sex="f" smoker="no"/>
  <person name="Angie Frost" age="33" sex="f" smoker="no"/>
  <person name="Betty Bates" age="33" sex="f" smoker="no"/>
  <person name="Connie Date" age="35" sex="f" smoker="no"/>
  <person name="Donna Finster" age="20" sex="f" smoker="no"/>
  <person name="Esther Gates" age="37" sex="f" smoker="no"/>
  <person name="Fanny Hill" age="33" sex="f" smoker="yes"/>
```

```
  <person name="Geta Iota" age="27" sex="f" smoker="no"/>
  <person name="Hillary Johnson" age="22" sex="f" smoker="no"/>
  <person name="Ingrid Kent" age="21" sex="f" smoker="no"/>
  <person name="Jill Larson" age="20" sex="f" smoker="no"/>
  <person name="Kim Mulrooney" age="41" sex="f" smoker="no"/>
  <person name="Lisa Nevins" age="21" sex="f" smoker="no"/>
</people>
```

Пример 7.7. Простое, но зависящее от конкретного входного документа преобразование в формат CSV

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template match="person">
       <xsl:value-of select="@name"/>,<xsl:text/>
       <xsl:value-of select="@age"/>,<xsl:text/>
       <xsl:value-of select="@sex"/>,<xsl:text/>
       <xsl:value-of select="@smoker"/>
       <xsl:text>&#xa;</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

Пример 7.8. Результат

```
Al Zehtooney,33,m,no
Brad York,38,m,yes
Charles Xavier,32,m,no
David Willimas,33,m,no
Edward Ulster,33,m,yes
Frank Townsend,35,m,no
Greg Sutter,40,m,no
...
```

Решение-то простое, но хотелось бы написать обобщенную таблицу стилей, которую было бы легко настроить на любое преобразование такого вида. В примерах 7.9 – 7.10 приведено подобное решение и показано, как его можно применить к файлу *people.xml*.

Пример 7.9. generic-attr-to-csv.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">
```

```
<xsl:param name="delimiter" select=" ',' "/>

<xsl:output method="text" />

<xsl:strip-space elements="*"/>

<xsl:template match="/">
  <xsl:for-each select="$columns">
    <xsl:value-of select="@name"/>
    <xsl:if test="position() != last()">
       <xsl:value-of select="$delimiter/>
    </xsl:if>
  </xsl:for-each>
  <xsl:text>&#xa;</xsl:text>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="/*/*">
  <xsl:variable name="row" select="."/>

  <xsl:for-each select="$columns">
    <xsl:apply-templates select="$row/@*[local-name(.)=current()/@attr]"
    mode="csv:map-value"/>
    <xsl:if test="position() != last()">
           <xsl:value-of select="$delimiter"/>
    </xsl:if>
  </xsl:for-each>

  <xsl:text>&#xa;</xsl:text>

</xsl:template>

<xsl:template match="@*" mode="map-value">
  <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.10. Применение обобщенного решения к файлу people.xml

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">

<xsl:import href="generic-attr-to-csv.xslt"/>

<!-- Определяем отображение атрибутов на колонки -->
```

```
<xsl:variable name="columns" select="document('')/*/csv:column"/>

<csv:column name="Name" attr="name"/>
<csv:column name="Age" attr="age"/>
<csv:column name="Gender" attr="sex"/>
<csv:column name="Smoker" attr="smoker"/>

<!-- Специальные отображения атрибутов -->

<xsl:template match="@sex" mode="csv:map-value">
    <xsl:choose>
           <xsl:when test=".='m'">male</xsl:when>
           <xsl:when test=".='f'">female</xsl:when>
           <xsl:otherwise>error</xsl:otherwise>
    </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Это таблично управляемое решение. В общей таблице стилей *generic-attr-to-csv.xslt* используется переменная, содержащая элементы `csv:column`, которые определены в импортирующей таблице, где нужно лишь перечислить элементы `csv:column` в том порядке, в котором они должны следовать в колонках выходного файла. Элементы `csv:column` определяют отображение между именованной колонкой и именем атрибута во входном XML-документе. Дополнительно импортирующая таблица стилей может транслировать значения некоторых атрибутов с помощью шаблона, соответствующего конкретному атрибуту. Этот шаблон запускается в режиме `csv:map-value`. В данном случае такой шаблон преобразует сокращенные обозначения пола в атрибуте `@sex` в полные слова. Часто используемые отображения такого рода можно поместить в третью таблицу стилей и тоже импортировать. Это решение удобно, потому что позволяет человеку, едва знакомому с XSLT, определить новое отображение XML в CSV. Дополнительный бонус в том, что в обобщенной таблице стилей определен параметр верхнего уровня, который позволяет заменить запятую в качестве разделителя другим символом.

Создание CSV-файла из плоского документа, в котором данные закодированы в элементах

Имеется плоский XML-файл, в котором элементы верхнего уровня отображаются на строки, а их дочерние элементы – на колонки.

Эта задача аналогична предыдущей, только на колонки отображаются не атрибуты, а элементы. Обобщенное решение приведено в примерах 7.11 – 7.14.

Пример 7.11. Документ, в котором люди описываются с помощью элементов

```
<people>
  <person>
```

```
    <name>Al Zehtooney</name>
    <age>33</age>
    <sex>m</sex>
    <smoker>no</smoker>
  </person>
  <person>
    <name>Brad York</name>
    <age>38</age>
    <sex>m</sex>
    <smoker>yes</smoker>
  </person>
  <person>
    <name>Charles Xavier</name>
    <age>32</age>
    <sex>m</sex>
    <smoker>no</smoker>
  </person>
  <person>
    <name>David Willimas</name>
    <age>33</age>
    <sex>m</sex>
    <smoker>no</smoker>
  </person>
...
</people>
```

Пример 7.12. generic-elem-to-csv.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">

<xsl:param name="delimiter" select=" ',' "/>

<xsl:output method="text" />

<xsl:strip-space elements="*"/>

<xsl:template match="/">
  <xsl:for-each select="$columns">
    <xsl:value-of select="@name"/>
    <xsl:if test="position() != last()">
        <xsl:value-of select="$delimiter"/>
    </xsl:if>
  </xsl:for-each>
```

```
  <xsl:text>&#xa;</xsl:text>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="/*/*">
  <xsl:variable name="row" select="."/>

  <xsl:for-each select="$columns">
    <xsl:apply-templates
       select="$row/*[local-name(.)=current()/@elem]" mode="csv:map-value"/>
    <xsl:if test="position() != last()">
       <xsl:value-of select="$delimiter"/>
    </xsl:if>
  </xsl:for-each>

  <xsl:text>&#xa;</xsl:text>

</xsl:template>

<xsl:template match="node()" mode="map-value">
  <xsl:value-of select="."/>
</xsl:template>
```

Пример 7.13. people-elem-to-csv.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">

<xsl:import href="generic-elem-to-csv.xslt"/>

<!-- Определяем отображение атрибутов на колонки -->
<xsl:variable name="columns" select="document('')/*/csv:column"/>

<csv:column name="Name" elem="name"/>
<csv:column name="Age" elem="age"/>
<csv:column name="Gender" elem="sex"/>
<csv:column name="Smoker" elem="smoker"/>

</xsl:stylesheet>
```

Пример 7.14. Результат

```
Name,Age,Gender,Smoker
Al Zehtooney,33,m,no
Brad York,38,m,yes
```

```
Charles Xavier,32,m,no
David Willimas,33,m,no
...
```

Более сложные отображения

В следующем примере нам предстоит выполнить отображение произвольной комбинации атрибутов и элементов на строки и колонки. Здесь порядок документа не так однозначно соответствует порядку строк и колонок. Кроме того, отображение может быть разреженным, то есть в CSV-файле оказывается много пустых значений.

Рассмотрим, к примеру, следующий XML-документ, представляющий отчет о расходах сотрудника, которого в ближайшем будущем ожидает увольнение.

```
<ExpenseReport statementNum="123">
  <Employee>
    <Name>Salvatore Mangano</Name>
    <SSN>999-99-9999</SSN>
    <Dept>XSLT Hacking</Dept>
    <EmpNo>1</EmpNo>
    <Position>Cook</Position>
    <Mangager>Big Boss O'Reilly</Mangager>
  </Employee>
  <PayPeriod>
    <From>1/1/02</From>
    <To>1/31/02</To>
  </PayPeriod>
  <Expenses>
    <Expense>
       <Date>12/20/01</Date>
       <Account>12345</Account>
       <Desc>Goofing off instead of going to confrence.</Desc>
       <Lodging>500.00</Lodging>
       <Transport>50.00</Transport>
       <Fuel>0</Fuel>
       <Meals>300.00</Meals>
       <Phone>100</Phone>
       <Entertainment>1000.00</Entertainment>
       <Other>300.00</Other>
    </Expense>
    <Expense>
       <Date>12/20/01</Date>
       <Account>12345</Account>
       <Desc>On the beach</Desc>
       <Lodging>500.00</Lodging>
       <Transport>50.00</Transport>
```

```
        <Fuel>0</Fuel>
        <Meals>200.00</Meals>
        <Phone>20</Phone>
        <Entertainment>300.00</Entertainment>
        <Other>100.00</Other>
      </Expense>
    </Expenses>
</ExpenseReport>
```

Предположим, что эти данные нужно импортировать в электронную таблицу, так чтобы после применения стилей получился результат, показанный на рис. 7.1.

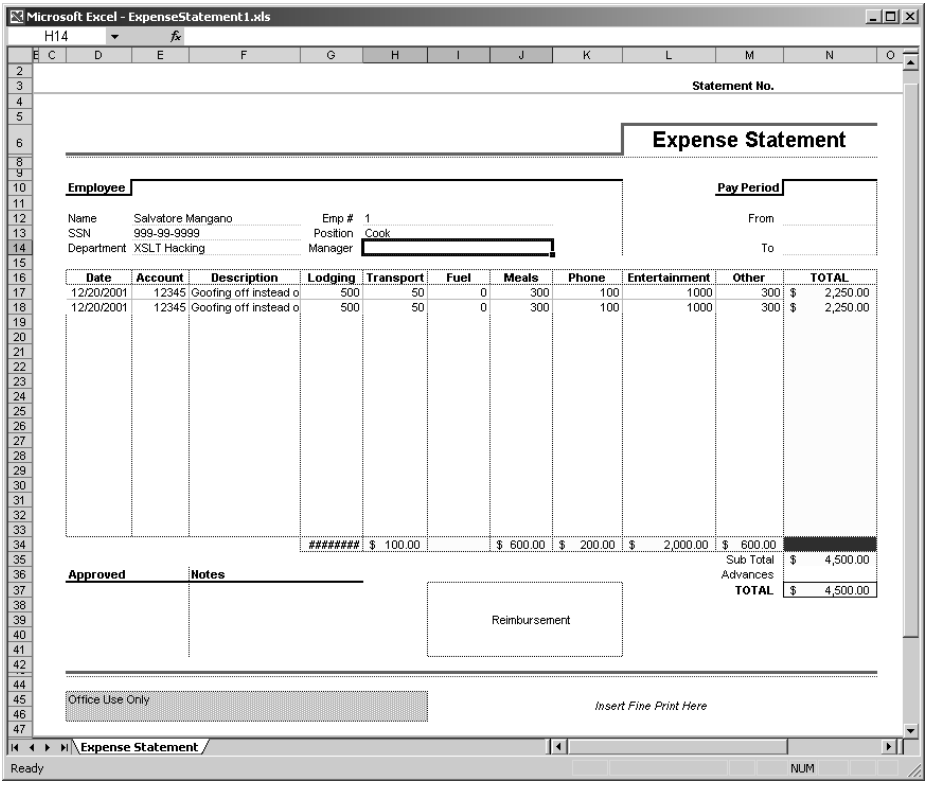


Рис. 7.1. Электронная таблица, содержащая отчет о расходах

Чтобы данные попали в нужные ячейки и осталось лишь применить стили, необходимо сгенерировать такой CSV-файл:

```
,,,,,,,,,,,Statement No.,123,

,,,,,,,,,,,Expense Statement,

,,,Employee,,,,,,,,,Pay Period,
```

```
,,,Name,Salvatore Mangano,,Emp #,1,,,,,From,1/1/02,
,,,SSN,999-99-9999,,Position,Cook,
,,,Department,XSLT Hacking,,,,,,,,To,1/31/02,

,,,Date,Account,Description,Lodging,Transport,Fuel,Meals,
Phone,Entertainment,Other,Total,

,,,12/20/01,12345,Goofing off instead of going to
confrence.,500.00,50.00,0,300.
00,100,1000.00,300.00,

,,,12/20/01,12345,On the
beach,500.00,50.00,0,200.00,20,300.00,100.00,Sub Total,
,,,Approved,,Notes,,,,,,,Advances,
,,,,,,,,,,,,Total,
```

Как видите, отображению XML на данные с разделителями не хватает единообразия, из-за которого решения предыдущих примеров были такими простыми. Я не хочу сказать, что для нужного преобразования нельзя создать таблицу стилей. Просто, если подойти к задаче прямолинейно, то таблица получится сложной и не пригодной для повторного использования.

Столкнувшись со сложным преобразованием, посмотрите, нельзя ли решить задачу, преобразовав документ сначала в промежуточную форму, а уже из нее – к нужному виду. Другими словами, попробуйте разбить трудную задачу на несколько более простых.

Взглянув на проблему с этой точки зрения, вы увидите, что задача преобразования XML в электронную таблицу на самом деле сводится к задаче распределения содержимого документа по ячейкам таблицы. Тогда в качестве промежуточной формы можно выбрать такую, где каждой ячейке соответствует отдельный элемент. Например, элемент, который должен поместить значение «foo» в ячейку A1 должен быть таким: `<cell col="A" row="1" value="foo"/>`. Ваша цель – создать таблицу стилей, которая отобразит каждый значимый элемент входного документа на элемент, соответствующий ячейке. Поскольку теперь беспокоиться о порядке уже не нужно, то отображение получается простым:

```
<xsl:template match="ExpenseReport">
    <c:cell col="M" row="3" value="Statement No."/>
    <c:cell col="N" row="3" value="{@statementNum}"/>
    <c:cell col="L" row="6" value="Expense Statement"/>
    <xsl:apply-templates/>
    <xsl:variable name="offset" select="count(Expenses/Expense)+18"/>
    <c:cell col="M" row="{$offset}" value="Sub Total"/>
    <c:cell col="D" row="{$offset + 1}" value="Approved"/>
    <c:cell col="F" row="{$offset + 1}" value="Notes"/>
    <c:cell col="M" row="{$offset + 1}" value="Advances"/>
```

```
  <c:cell col="M" row="{$offset + 2}" value="Total"/>
</xsl:template>

<xsl:template match="Employee">
  <c:cell col="D" row="10" value="Employee"/>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="Employee/Name">
  <c:cell col="D" row="12" value="Name"/>
  <c:cell col="E" row="12" value="{.}"/>
</xsl:template>

<xsl:template match="Employee/SSN">
  <c:cell col="D" row="13" value="SSN"/>
  <c:cell col="E" row="13" value="{.}"/>
</xsl:template>

<xsl:template match="Employee/Dept">
  <c:cell col="D" row="14" value="Department"/>
  <c:cell col="E" row="14" value="{.}"/>
</xsl:template>

<xsl:template match="Employee/EmpNo">
  <c:cell col="G" row="12" value="Emp #"/>
  <c:cell col="H" row="12" value="{.}"/>
</xsl:template>

<xsl:template match="Employee/Position">
  <c:cell col="G" row="13" value="Position"/>
  <c:cell col="H" row="13" value="{.}"/>
</xsl:template>

<xsl:template match="Employee/Manager">
  <c:cell col="G" row="14" value="Manager"/>
  <c:cell col="H" row="14" value="{.}"/>
</xsl:template>

<xsl:template match="PayPeriod">
  <c:cell col="M" row="10" value="Pay Period"/>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="PayPeriod/From">
```

```
  <c:cell col="M" row="12" value="From"/>
  <c:cell col="N" row="12" value="{.}"/>
</xsl:template>

<xsl:template match="PayPeriod/To">
  <c:cell col="M" row="14" value="To"/>
  <c:cell col="N" row="14" value="{.}"/>
</xsl:template>

<xsl:template match="Expenses">
  <c:cell col="D" row="16" value="Date"/>
  <c:cell col="E" row="16" value="Account"/>
  <c:cell col="F" row="16" value="Description"/>
  <c:cell col="G" row="16" value="Lodging"/>
  <c:cell col="H" row="16" value="Transport"/>
  <c:cell col="I" row="16" value="Fuel"/>
  <c:cell col="J" row="16" value="Meals"/>
  <c:cell col="K" row="16" value="Phone"/>
  <c:cell col="L" row="16" value="Entertainment"/>
  <c:cell col="M" row="16" value="Other"/>
  <c:cell col="N" row="16" value="Total"/>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="Expenses/Expense">
  <xsl:apply-templates>
    <xsl:with-param name="row" select="position()+16"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="Expense/Date">
  <xsl:param name="row"/>
  <c:cell col="D" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Account">
  <xsl:param name="row"/>
  <c:cell col="E" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Desc">
  <xsl:param name="row"/>
  <c:cell col="F" row="{$row}" value="{.}"/>
</xsl:template>
```

```
<xsl:template match="Expense/Lodging">
  <xsl:param name="row"/>
  <c:cell col="G" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Transport">
  <xsl:param name="row"/>
  <c:cell col="H" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Fuel">
  <xsl:param name="row"/>
  <c:cell col="I" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Meals">
  <xsl:param name="row"/>
  <c:cell col="J" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Phone">
  <xsl:param name="row"/>
  <c:cell col="K" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Entertainment">
  <xsl:param name="row"/>
  <c:cell col="L" row="{$row}" value="{.}"/>
</xsl:template>

<xsl:template match="Expense/Other">
  <xsl:param name="row"/>
  <c:cell col="M" row="{$row}" value="{.}"/>
</xsl:template>
```

У кодирования значений ячеек атрибутами есть важное преимущество – вы получаете возможность воспользоваться шаблонами, отображающими атрибут на значение, что приводит к очень компактной схеме трансляции. В этой таблице стилей производятся преобразования двух видов. Первое – абсолютное. например, имя работника отображается на ячейку E12. Второе – относительное; каждая статья расходов отображается на некоторую строку относительно строки 16 в зависимости от позиции в исходном документе.

Применив эту таблицу стилей к исходному документу, получим следующий результат:

```
<c:cells xmlns:c="http://www.ora.com/XSLTCookbook/namespaces/cells" >
  <c:cell col="M" row="3" value="Statement No."/>
  <c:cell col="N" row="3" value="123"/>
  <c:cell col="L" row="6" value="Expense Statement"/>
  <c:cell col="D" row="10" value="Employee"/>
  <c:cell col="D" row="12" value="Name"/>
  <c:cell col="E" row="12" value="Salvatore Mangano"/>
  <c:cell col="D" row="13" value="SSN"/>
  <c:cell col="E" row="13" value="999-99-9999"/>
  <c:cell col="D" row="14" value="Department"/>
  <c:cell col="E" row="14" value="XSLT Hacking"/>
  <c:cell col="G" row="12" value="Emp #"/>
  <c:cell col="H" row="12" value="1"/>
  <c:cell col="G" row="13" value="Position"/>
  <c:cell col="H" row="13" value="Cook"/>
  <c:cell col="G" row="14" value="Manager"/>
  <c:cell col="H" row="14" value="Big Boss O'Reilly"/>
  <c:cell col="M" row="10" value="Pay Period"/>
  <c:cell col="M" row="12" value="From"/>
  <c:cell col="N" row="12" value="1/1/02"/>
  <c:cell col="M" row="14" value="To"/>
  <c:cell col="N" row="14" value="1/31/02"/>
  <c:cell col="D" row="16" value="Date"/>
  <c:cell col="E" row="16" value="Account"/>
  <c:cell col="F" row="16" value="Description"/>
  <c:cell col="G" row="16" value="Lodging"/>
  <c:cell col="H" row="16" value="Transport"/>
  <c:cell col="I" row="16" value="Fuel"/>
  <c:cell col="J" row="16" value="Meals"/>
  <c:cell col="K" row="16" value="Phone"/>
  <c:cell col="L" row="16" value="Entertainment"/>
  <c:cell col="M" row="16" value="Other"/>
  <c:cell col="N" row="16" value="Total"/>
  <c:cell col="D" row="18" value="12/20/01"/>
  <c:cell col="E" row="18" value="12345"/>
  <c:cell col="F" row="18" value="Goofing off instead of going to confrence."/>
  <c:cell col="G" row="18" value="500.00"/>
  <c:cell col="H" row="18" value="50.00"/>
  <c:cell col="I" row="18" value="0"/>
  <c:cell col="J" row="18" value="300.00"/>
  <c:cell col="K" row="18" value="100"/>
  <c:cell col="L" row="18" value="1000.00"/>
  <c:cell col="M" row="18" value="300.00"/>
  <c:cell col="D" row="20" value="12/20/01"/>
```

```
  <c:cell col="E" row="20" value="12345"/>
  <c:cell col="F" row="20" value="On the beach"/>
  <c:cell col="G" row="20" value="500.00"/>
  <c:cell col="H" row="20" value="50.00"/>
  <c:cell col="I" row="20" value="0"/>
  <c:cell col="J" row="20" value="200.00"/>
  <c:cell col="K" row="20" value="20"/>
  <c:cell col="L" row="20" value="300.00"/>
  <c:cell col="M" row="20" value="100.00"/>
  <c:cell col="M" row="20" value="Sub Total"/>
  <c:cell col="D" row="21" value="Approved"/>
  <c:cell col="F" row="21" value="Notes"/>
  <c:cell col="M" row="21" value="Advances"/>
  <c:cell col="M" row="22" value="Total"/>
</c:cells>
```

Конечно, это еще не то, что мы хотим получить. Однако нетрудно видеть, что после сортировки ячеек сначала по `@row`, а потом по `@col` преобразование в формат с разделителями полей окажется совсем простым. На самом деле, если вы готовы прибегнуть к функции расширения `node-set`, определенной в проекте EXSLT, то получить желаемый результат можно и за один проход. Заметим также, что преобразование элементов `cell` в последовательность полей, разделенных запятыми, имеет общий характер, поэтому его можно будет использовать повторно для других сложных преобразований XML. См. примеры 7.17 и 7.16.

Пример 7.15. Обобщенная таблица стилей cells-to-comma-delimited.xslt

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:c="http://www.ora.com/XSLTCookbook/namespaces/cells"
  xmlns:exsl="http://exslt.org/common" extension-element-prefixes="exsl">

  <xsl:output method="text"/>

  <!-- Отображаем буквы, обозначающие колонки, на числа -->
  <xsl:variable name="columns" select=" '_ABCDEFGHIJKLMNOPQRSTUVWXYZ' "/>

  <xsl:template match="/">

  <!-- Запоминаем ячейки в переменной -->
    <xsl:variable name="cells">
      <xsl:apply-templates/>
  </xsl:variable>

  <!-- Сортируем в порядке строка-колонка -->
```

```
 <xsl:variable name="cells-sorted">
    <xsl:for-each select="exsl:node-set($cells)/c:cell">
        <xsl:sort select="@row" data-type="number"/>
        <xsl:sort select="@col" data-type="text"/>
        <xsl:copy-of select="."/>
    </xsl:for-each>
 </xsl:variable>

 <xsl:apply-templates select="exsl:node-set($cells-sorted)/c:cell"/>

</xsl:template>

  <xsl:template match="c:cell">
    <xsl:choose>
      <!-- Обнаруживаем смену строки -->
      <xsl:when test="preceding-sibling::c:cell[1]/@row != @row">
        <!-- Вычисляем, сколько строк пропустить -->
        <xsl:variable name="skip-rows">
          <xsl:choose>
            <xsl:when test="preceding-sibling::c:cell[1]/@row">
              <xsl:value-of
                  select="@row - preceding-sibling::c:cell[1]/@row"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="@row - 1"/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:variable>
        <xsl:call-template name="skip-rows">
          <xsl:with-param name="skip" select="$skip-rows"/>
        </xsl:call-template>

        <xsl:variable name="current-col"
               select="string-length(substring-before($columns,@col))"/>
        <xsl:call-template name="skip-cols">
          <xsl:with-param name="skip" select="$current-col - 1"/>
        </xsl:call-template>
        <xsl:value-of select="@value"/>,<xsl:text/>
      </xsl:when>

      <xsl:otherwise>
        <!-- Вычисляем, сколько колонок пропустить -->
        <xsl:variable name="skip-cols">
          <xsl:variable name="current-col"
               select="string-length(substring-before($columns,@col))"/>
```

```
            <xsl:choose>
              <xsl:when test="preceding-sibling::c:cell[1]/@col">
                <xsl:variable name="prev-col"
                 select="string-length(substring-before($columns,
                           preceding-sibling::c:cell[1]/@col))"/>
                <xsl:value-of select="$current-col - $prev-col - 1"/>
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="$current-col - 1"/>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>

          <xsl:call-template name="skip-cols">
            <xsl:with-param name="skip" select="$skip-cols"/>
          </xsl:call-template>
          <!-- Output the value of the cell and a comma -->
          <xsl:value-of select="@value"/>,<xsl:text/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:template>

<!-- Вставляем пустые строки, если строки таблицы не соседние -->
<xsl:template name="skip-rows">
  <xsl:param name="skip"/>
  <xsl:choose>
    <xsl:when test="$skip > 0">
      <xsl:text>&#xa;</xsl:text>
      <xsl:call-template name="skip-rows">
        <xsl:with-param name="skip" select="$skip - 1"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
</xsl:template>

<!-- Вставляем дополнительные запятые, если колонки не соседние -->
<xsl:template name="skip-cols">
  <xsl:param name="skip"/>
  <xsl:choose>
    <xsl:when test="$skip > 0">
      <xsl:text>,</xsl:text>
      <xsl:call-template name="skip-cols">
        <xsl:with-param name="skip" select="$skip - 1"/>
```

```
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.16. Таблица стилей expense-to-delimited.xslt для конкретного приложения

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:c="http://www.ora.com/XSLTCookbook/namespaces/cells"
  xmlns:exsl="http://exslt.org/common" extension-element-prefixes="exsl">

  <xsl:include href="cells-to-comma-delimited.xslt"/>

  <xsl:template match="ExpenseReport">
    <c:cell col="M" row="3" value="Statement No."/>
    <c:cell col="N" row="3" value="{@statementNum}"/>
    <c:cell col="L" row="6" value="Expense Statement"/>
    <xsl:apply-templates/>
    <xsl:variable name="offset" select="count(Expenses/Expense)+18"/>
    <c:cell col="M" row="{$offset}" value="Sub Total"/>
    <c:cell col="D" row="{$offset + 1}" value="Approved"/>
    <c:cell col="F" row="{$offset + 1}" value="Notes"/>
    <c:cell col="M" row="{$offset + 1}" value="Advances"/>
    <c:cell col="M" row="{$offset + 2}" value="Total"/>
  </xsl:template>

  <xsl:template match="Employee">
    <c:cell col="D" row="10" value="Employee"/>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="Employee/Name">
    <c:cell col="D" row="12" value="Name"/>
    <c:cell col="E" row="12" value="{.}"/>
  </xsl:template>

<!-- ... -->
<!-- Опущенный код такой же, как в исходной таблице стилей выше -->
<!-- ... -->

</xsl:stylesheet>
```

В обобщенной таблице стилей *cells-to-comma-delimited.xslt* ячейки, порожденные таблицей, специфичной для приложения, запоминаются в переменной и сортируются. Затем эти ячейки преобразуются в файл с полями, разделенными запятыми. Для этого каждый элемент cell сравнивается со своим предшественником после сортировки. Если предшественник относится к другой строке, то нужно вывести один или несколько символов новой строки. Если же предшественник принадлежит не соседней колонке, то следует вывести одну или несколько дополнительных запятых. Необходимо рассмотреть также случай, когда первая строка или первая колонка в строке не являются соответственно первой строкой или колонкой в электронной таблице. Обработав все эти детали, останется только вывести значение ячейки и запятую.

XSLT 2.0

Новый атрибут `separator` элемента `xsl:value-of` заметно упрощает вывод текста с разделителями. Когда элементу `xsl:value-of` передана последовательность, он сериализует ее; при этом, если задан атрибут separator, то после каждого члена последовательности, кроме последнего вставляется значение этого атрибута – разделитель. Разделитель может быть литералом или вычисляемым выражением. В следующем примере я воспользовался этой возможностью, чтобы упростить код, который выводит имена колонок. Кроме того, средства XPath 2.0 позволили мне обобщить решение так, чтобы одну и ту же базовую таблицу стилей можно было применять к XML-документам, в которых данные закодированы в элементах или атрибутах. Наконец, для кодирования отображения XML на CSV я использую литеральные последовательности, а не внедренный в таблицу стилей XML-код. Все это сделано просто для иллюстрации гибкости XSLT 2.0, без намека на то, что одна техника предпочтительнее другой (см. примеры 7.17 и 7.18).

Пример 7.17. Обобщенная таблица стилей cells-to-comma-delimited.xslt

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2004/10/xpath-functions"
xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">

<xsl:param name="delimiter" select=" ',' "/>

<!-- Должно быть переопределено в импортирующей таблице стилей -->
<xsl:variable name="columns" select="()" as="xs:string*"/>
<xsl:variable name="nodeNames" select="$columns" as="xs:string*"/>

<xsl:output method="text" />

<xsl:strip-space elements="*"/>

<xsl:template match="/">
```

```
  <!-- Здесь используется новый атрибут value-of -->
  <xsl:value-of select="$columns" separator="{$delimiter}"/>
  <xsl:text>&#xa;</xsl:text>
  <xsl:apply-templates mode="csv:map-row"/>
</xsl:template>

<xsl:template match="/*/*" mode="csv:map-row" name="csv:map-row">

  <xsl:param name="elemOrAttr" select=" 'elem' " as="xs:string"/>

  <xsl:variable name="row" select="." as="node()"/>

  <xsl:for-each select="$nodeNames">
    <xsl:apply-templates select="if ($elemOrAttr eq 'elem')
                                 then $row/*[local-name(.) eq current()]
                                 else $row/@*[local-name(.) eq current()]"
                         mode="csv:map-value"/>
    <xsl:value-of select="if (position() ne last()) then $delimiter else ()"/>
  </xsl:for-each>

  <xsl:text>&#xa;</xsl:text>

</xsl:template>

<xsl:template match="node()" mode="csv:map-value">
  <xsl:value-of select="."/>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.18. Таблица стилей expense-to-delimited.xslt для конкретного приложения

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:csv="http://www.ora.com/XSLTCookbook/namespaces/csv">

<xsl:import href="toCSV.xslt"/>

<!-- Определяем отображение узлов на колонки -->
<xsl:variable name="columns" select="'Name', 'Age', 'Gender', 'Smoker'"
              as="xs:string*"/>
<xsl:variable name="nodeNames" select="'name', 'age', 'sex', 'smoker'"
              as="xs:string*"/>
```

```
<!-- Переключаем обработку по умолчанию с элементов на атрибуты -->
<xsl:template match="/*/*" mode="csv:map-row">
    <xsl:call-template name="csv:map-row">
            <xsl:with-param    name="elemOrAttr"    select="    'attr'    "/>
    </xsl:call-template>
</xsl:template>

<!-- Обрабатываем специальные отображения атрибутов -->

<xsl:template match="@sex" mode="csv:map-value">
    <xsl:choose>
            <xsl:when test=".='m'">male</xsl:when>
            <xsl:when test=".='f'">female</xsl:when>
            <xsl:otherwise>error</xsl:otherwise>
    </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Большинство преобразований XML в формат с разделителями полей для знатоков XSLT не представляют сложности. Ценность предыдущих примеров в демонстрации того, как задачу можно разбить на две части: повторно используемую, для написания которой требуется хорошее знание XSLT, и прикладную, которая доступна менее опытным пользователям, если они хорошо понимают принятые соглашения.

Ценность такого подхода заключается в том, что он позволяет выполнять полезную работу людям, не владеющим XSLT в полной мере. Предположим, например, что нужно конвертировать большую базу XML-документов в формат с разделителями полей, причем, как обычно, это следовало сделать еще вчера. Показать кому-то, как пользоваться общим решением, куда проще, чем обучить его XSLT в объеме, позволяющем составлять специализированные таблицы стилей.

7.3. Создание отчета с несколькими колонками

Задача

Требует представить данные в виде таблицы с несколькими колонками.

Решение

Есть две разновидности отображения XML в многоколонную таблицу. В первом случае различные элементы или атрибуты отображаются на разные колонки, во втором отображение элемента определяется его относительной позицией.

Прежде чем переходить к изучению этих двух вариантов, напишем общий шаблон для выравнивания текста в колонке фиксированной ширины. Такую утилиту (см. пример 7.19) можно написать на базе шаблона `str:dup` из рецепта 2.5.

Пример 7.19. Общий шаблон выравнивания текста text.justify.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
  xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text"
  extension-element-prefixes="text">

<xsl:include href="../strings/str.dup.xslt"/>

<xsl:template name="text:justify">
  <xsl:param name="value" />
  <xsl:param name="width" select="10"/>
  <xsl:param name="align" select=" 'left' "/>

  <!-- Обрезать, если строка слишком длинная -->
  <xsl:variable name="output" select="substring($value,1,$width)"/>

  <xsl:choose>
   <xsl:when test="$align = 'left'">
    <xsl:value-of select="$output"/>
    <xsl:call-template name="str:dup">
       <xsl:with-param name="input" select=" ' ' "/>
       <xsl:with-param name="count"
            select="$width - string-length($output)"/>
    </xsl:call-template>
   </xsl:when>
   <xsl:when test="$align = 'right'">
    <xsl:call-template name="str:dup">
       <xsl:with-param name="input" select=" ' ' "/>
       <xsl:with-param name="count"
            select="$width - string-length($output)"/>
    </xsl:call-template>
    <xsl:value-of select="$output"/>
   </xsl:when>
   <xsl:when test="$align = 'center'">
    <xsl:call-template name="str:dup">
          <xsl:with-param name="input" select=" ' ' "/>
          <xsl:with-param name="count"
               select="floor(($width - string-length($output)) div 2)"/>
    </xsl:call-template>
    <xsl:value-of select="$output"/>
```

```
      <xsl:call-template name="str:dup">
             <xsl:with-param name="input" select=" ' ' "/>
             <xsl:with-param name="count"
               select="ceiling(($width - string-length($output)) div 2)"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>INVALID ALIGN</xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  </xsl:stylesheet>
```

Имея такой шаблон, для создания поколонного отчета достаточно задать лишь порядок и разметку колонок. В примерах 7.20 и 7.21 это сделано для файла people.xml, где данные о сотрудниках представлены в виде атрибутов. Аналогично можно было бы поступить для файла people_elem.xml, где данные закодированы в элементах.

Пример 7.20. people-to-columns.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="htp://www.w3.org/1999/XSL/Transform
  xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
  xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text"

<xsl:include href="text.jstify.xslt"/>

<xsl:output method="text" />

<xsl:strip-space elements="*"/>

<xsl:template match="people">
Name                Age    Sex     Smoker
-------------------|------|-------|----------
<xsl:apply-templates>

<xsl:template match="person">
    <xsl:call-template name="text:justify">
    <xsl:with-param name="value" select="@name"/>
    <xsl:with-param name="width" select="20"/>
  <xsl:call-template>
<xsl:text>|</xsl:text>
    <xsl:call-template name="text:justify">
    <xsl:with-param name="value" select="@age"/>
    <xsl:with-param name="width" select="6"/>
    <xsl:with-param name="align" select=" 'right' "/>
  <xsl:call-template>
```

```
<xsl:text>|</xsl:text>
    <xsl:call-template name="text:justify">
    <xsl:with-param name="value" select="@sex"/>
    <xsl:with-param name="width" select="6"/>
    <xsl:with-param name="align" select=" 'center' "/>
  <xsl:call-template>
<xsl:text>|</xsl:text>
    <xsl:call-template name="text:justify">
    <xsl:with-param name="value" select="@smoker"/>
    <xsl:with-param name="width" select="9"/>
    <xsl:with-param name="align" select=" 'center' "/>
  <xsl:call-template>
  <xsl:text>
</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.21. Результат

```
Name                |  Age  |  Sex  |  Smoker
Al Zehtooney        |    33 |   m   |   no
Brad York           |    38 |   m   |  yes
Charles Xavier      |    32 |   m   |   no
David Williams      |    33 |   m   |   no
Edward Ulster       |    33 |   m   |  yes
Frank Townsend      |    35 |   m   |   no
Greg Sutter         |    40 |   m   |   no
Harry Rogers        |    37 |   m   |   no
John Quincy         |    43 |   m   |  yes
Kent Peterson       |    31 |   m   |   no
Larry Newell        |    23 |   m   |   no
Max Milton          |    22 |   m   |   no
Norman Lamagna      |    30 |   m   |   no
Ollie Kensinton     |    44 |   m   |   no
John Frank          |    24 |   m   |   no
Mary Williams       |    33 |   f   |   no
Jane Frank          |    38 |   f   |  yes
Jo Peterson         |    32 |   f   |   no
Angie Frost         |    33 |   f   |   no
Betty Bates         |    33 |   f   |   no
Connie Date         |    35 |   f   |   no
Donna Finster       |    20 |   f   |   no
Esther Gates        |    37 |   f   |   no
Fanny Hill          |    33 |   f   |  yes
Geta Iota           |    27 |   f   |   no
Hillary Johnson     |    22 |   f   |   no
```

```
Ingrid Kent          |    21|   f   |   no
Jill Larson          |    20|   f   |   no
Kim Mulrooney        |    41|   f   |   no
Lisa Nevins          |    21|   f   |   no
```

Чтобы преобразовать данные на основе позиции в документе, к задаче нужно подойти несколько иначе. Во-первых, решите, сколько должно быть колонок. Можно задать число колонок с помощью параметра, а число строк вычислить, исходя из количества элементов. А можно задать число строк, тогда количество колонок будет переменным. Во-вторых, определите, как позиция элемента будет отображаться на номер колонки. Есть два распространенных способа: сначала по строкам или сначала по колонкам. В первом случае первый элемент помещается в первую колонку, второй – во вторую и так далее, пока колонки не кончатся; в этот момент начинается новая строка. Во втором случае первые (N div число-колонок) элементов помещаются в первую колонку, следующие (N div число-колонок) элементов – во вторую и т.д. Иными словами, строки и колонки меняются местами – транспонируются.

В примере 7.22 приведены два шаблона – для вывода отчета по строкам и по колонкам.

Пример 7.22. text.matrix.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text"
  extension-element-prefixes="text">

<xsl:output method="text"/>

<xsl:include href="text.justify.xslt"/>

<xsl:template name="text:row-major">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="num-cols" select="2"/>
  <xsl:param name="width" select="10"/>
  <xsl:param name="align" select=" 'left' "/>
  <xsl:param name="gutter" select=" ' ' "/>

  <xsl:if test="$nodes">
    <xsl:call-template name="text:row">
       <xsl:with-param name="nodes"
           select="$nodes[position() &lt;= $num-cols]"/>
       <xsl:with-param name="width" select="$width"/>
       <xsl:with-param name="align" select="$align"/>
       <xsl:with-param name="gutter" select="$gutter"/>
    </xsl:call-template>
```

```
    <!-- обработать остальные строки -->
    <xsl:call-template name="text:row-major">
       <xsl:with-param name="nodes"
            select="$nodes[position() > $num-cols]"/>
       <xsl:with-param name="num-cols" select="$num-cols"/>
       <xsl:with-param name="width" select="$width"/>
       <xsl:with-param name="align" select="$align"/>
       <xsl:with-param name="gutter" select="$gutter"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="text:col-major">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="num-cols" select="2"/>
  <xsl:param name="width" select="10"/>
  <xsl:param name="align" select=" 'left' "/>
  <xsl:param name="gutter" select=" ' ' "/>

  <xsl:if test="$nodes">
    <xsl:call-template name="text:row">
        <xsl:with-param name="nodes"
           select="$nodes[(position() - 1) mod
                  ceiling(last() div $num-cols) = 0]"/>
        <xsl:with-param name="width" select="$width"/>
        <xsl:with-param name="align" select="$align"/>
        <xsl:with-param name="gutter" select="$gutter"/>
    </xsl:call-template>

    <!-- Обработать остальные колонки -->
    <xsl:call-template name="text:col-major">
        <xsl:with-param name="nodes"
           select="$nodes[(position() - 1) mod
                  ceiling(last() div $num-cols) != 0]"/>
        <xsl:with-param name="num-cols" select="$num-cols"/>
        <xsl:with-param name="width" select="$width"/>
        <xsl:with-param name="align" select="$align"/>
        <xsl:with-param name="gutter" select="$gutter"/>
    </xsl:call-template>
  </xsl:if>

</xsl:template>

<xsl:template name="text:row">
```

```
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="width" select="10"/>
  <xsl:param name="align" select=" 'left' "/>
  <xsl:param name="gutter" select=" ' ' "/>

  <xsl:for-each select="$nodes">
    <xsl:call-template name="text:justify">
       <xsl:with-param name="value" select="."/>
       <xsl:with-param name="width" select="$width"/>
       <xsl:with-param name="align" select="$align"/>
    </xsl:call-template>
    <xsl:value-of select="$gutter"/>
  </xsl:for-each>

  <xsl:text>&#xa;</xsl:text>

</xsl:template>

</xsl:stylesheet>
```

Этими шаблонами можно воспользоваться, как показано в примерах 7.23 – 7.25.

Пример 7.23. Входные данные

```
<numbers>
    <number>10</number>
    <number>3.5</number>
    <number>4.44</number>
    <number>77.7777</number>
    <number>-8</number>
    <number>1</number>
    <number>444</number>
    <number>1.1234</number>
    <number>7.77</number>
    <number>3.1415927</number>
    <number>10</number>
    <number>9</number>
    <number>8</number>
    <number>7</number>
    <number>666</number>
    <number>5555</number>
    <number>-4444444</number>
    <number>22.33</number>
    <number>18</number>
    <number>36.54</number>
    <number>43</number>
```

```
    <number>99999</number>
    <number>999999</number>
    <number>9999999</number>
    <number>32</number>
    <number>64</number>
    <number>-64.0001</number>
</numbers>
```

Пример 7.24. Таблица стилей

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text">

<xsl:output method="text" />

<xsl:include href="text.matrix.xslt"/>

<xsl:template match="numbers">
Пять колонок чисел, выведенных по строкам:
<xsl:text/>
<xsl:call-template name="text:row-major">
    <xsl:with-param name="nodes" select="number"/>
    <xsl:with-param name="align" select=" 'right' "/>
    <xsl:with-param name="num-cols" select="5"/>
    <xsl:with-param name="gutter" select=" ' | ' "/>
</xsl:call-template>

Пять колонок чисел, выведенных по колонкам:
<xsl:text/>
<xsl:call-template name="text:col-major">
    <xsl:with-param name="nodes" select="number"/>
    <xsl:with-param name="align" select=" 'right' "/>
    <xsl:with-param name="num-cols" select="5"/>
    <xsl:with-param name="gutter" select=" ' | ' "/>
</xsl:call-template>

</xsl:template>

</xsl:stylesheet>
```

Пример 7.25. Результат

```
Пять колонок чисел, выведенных по строкам:
       10 |          3.5 |         4.44 |      77.7777 |           -8 |
        1 |          444 |       1.1234 |         7.77 |    3.1415927 |
```

```
        10 |            9 |           8 |           7 |         666 |
      5555 |    -4444444 |       22.33 |          18 |       36.54 |
        43 |        99999 |      999999 |     9999999 |          32 |
        64 |    -64.0001 |

Пять колонок чисел, выведенных по колонкам:
        10 |          444 |           8 |          18 |          32 |
       3.5 |       1.1234 |           7 |       36.54 |          64 |
      4.44 |         7.77 |         666 |          43 |    -64.0001 |
     77.7777 |    3.1415927 |        5555 |       99999 |
        -8 |           10 |    -4444444 |      999999 |
         1 |            9 |       22.33 |     9999999 |
```

XSLT 2.0

В XSLT 2.0 можно превратить шаблон `text:justify` в функцию и сделать его более компактным за счет новых средств XPath 2.0 – это, пожалуй, основное улучшение, которого можно добиться. С помощью функции `string-join` в сочетании с выражением `for` создадим функцию `dup`, которая будет вставлять нужное для выравнивания число пробелов. Функцию `text:justify` можно перегрузить, чтобы добиться эффекта параметра по умолчанию, определяющего способ выравнивания:

```
<xsl:function name="text:dup" as="xs:string">
  <xsl:param name="input" as="xs:string"/>
  <xsl:param name="count" as="xs:integer"/>
  <xsl:sequence  select="string-join(for $i in 1 to $count return $input, '')"/>
</xsl:function>

<xsl:function name="text:justify" as="xs:string">
  <xsl:param name="value" as="xs:string"/>
  <xsl:param name="width" as="xs:integer" />
  <xsl:sequence select="text:justify($value, $width, 'left')"/>
</xsl:function>

<xsl:function name="text:justify" as="xs:string">
  <xsl:param name="value" as="xs:string"/>
  <xsl:param name="width" as="xs:integer" />
  <xsl:param name="align" as="xs:string" />

  <!-- Обрезать, если строка слишком длинная -->
  <xsl:variable name="output"
                select="substring($value,1,$width)" as="xs:string"/>
  <xsl:variable name="offset"
                select="$width - string-length($output)" as="xs:integer"/>
  <xsl:choose>
    <xsl:when test="$align = 'left'">
```

```
        <xsl:value-of select="concat($output, text:dup(' ', $offset))"/>
      </xsl:when>
      <xsl:when test="$align = 'right'">
        <xsl:value-of select="concat(text:dup(' ', $offset), $output)"/>
      </xsl:when>
      <xsl:when test="$align = 'center'">
        <xsl:variable name="before" select="$offset idiv 2"/>
        <xsl:variable name="after" select="$before + $offset mod 2"/>
        <xsl:value-of select="concat(text:dup(' ', $before),
                                     $output,text:dup(' ', $after))"/>
      </xsl:when>
      <xsl:otherwise>INVALID ALIGN</xsl:otherwise>
    </xsl:choose>
  </xsl:function>
```

Обсуждение

Задача преобразования данных, закодированных в виде элементов или атрибутов, в форму таблицы с несколькими колонками структурно похожа на задачу преобразования в формат с разделителями полей, которая обсуждалась в рецепте 7.2. Основное различие в том, что во втором случае данные готовятся для машинной обработки, а в первом – для человека. В некоторых отношениях человек более привередлив, чем машина, особенно когда речь идет о выравнивании и других способах оформления, облегчающих визуальное восприятие. Можно было бы применить тот же таблично управляемый подход, что и для формата с разделителями полей, но для правильного форматирования придется задать больше информации о каждой колонке. В примерах 7.26 – 7.28 приведено такое решение для случая, когда данные кодируются в атрибутах.

Пример 7.26. generic-attr-to-columns.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
    xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text">

<xsl:include href="text.justify.xslt"/>

<xsl:param name="gutter" select=" ' ' "/>

<xsl:output method="text"/>

<xsl:strip-space elements="*"/>

<xsl:variable name="columns" select="/.."/>

<xsl:template match="/">
```

```
    <xsl:for-each select="$columns">
      <xsl:call-template name="text:justify" >
         <xsl:with-param name="value" select="@name"/>
         <xsl:with-param name="width" select="@width"/>
         <xsl:with-param name="align" select=" 'left' "/>
      </xsl:call-template>
      <xsl:value-of select="$gutter"/>
    </xsl:for-each>
    <xsl:text>&#xa;</xsl:text>
    <xsl:for-each select="$columns">
      <xsl:call-template name="str:dup">
          <xsl:with-param name="input" select=" '-' "/>
          <xsl:with-param name="count" select="@width"/>
      </xsl:call-template>
      <xsl:call-template name="str:dup">
          <xsl:with-param name="input" select=" '-' "/>
          <xsl:with-param name="count" select="string-length($gutter)"/>
      </xsl:call-template>
    </xsl:for-each>
    <xsl:text>&#xa;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="/*/*">
    <xsl:variable name="row" select="."/>

    <xsl:for-each select="$columns">
      <xsl:variable name="value">
        <xsl:apply-templates
        select="$row/@*[local-name(.)=current()/@attr]"
                mode="text:map-col-value"/>
      </xsl:variable>
      <xsl:call-template name="text:justify" >
          <xsl:with-param name="value" select="$value"/>
          <xsl:with-param name="width" select="@width"/>
          <xsl:with-param name="align" select="@align"/>
      </xsl:call-template>
      <xsl:value-of select="$gutter"/>
    </xsl:for-each>

    <xsl:text>&#xa;</xsl:text>

  </xsl:template>

  <xsl:template match="@*" mode="text:map-col-value">
```

```
  <xsl:value-of select="."/>
</xsl:template>
```

Пример 7.27. people-to-cols-using-generic.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
    xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text">

<xsl:import href="generic-attr-to-columns.xslt"/>

<!-- Определяем отображение атрибутов на колонки -->
<xsl:variable name="columns" select="document('')/*/text:column"/>

<text:column name="Name" width="20" align="left" attr="name"/>
<text:column name="Age" width="6" align="right" attr="age"/>
<text:column name="Gender" width="6" align="left" attr="sex"/>
<text:column name="Smoker" width="6" align="left" attr="smoker"/>

<!-- Обрабатываем специальные отображения атрибутов -->

<xsl:template match="@sex" mode="text:map-col-value">
  <xsl:choose>
    <xsl:when test=".='m''">male</xsl:when>
    <xsl:when test=".='f'">female</xsl:when>
    <xsl:otherwise>ошибка</xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.28. Результат (параметр gutter = " | ")

```
Name                | Age     | Gender  | Smoker   |
————————————————————————————————————————————————
Al Zehtooney        |      33 | male    | no       |
Brad York           |      38 | male    | yes      |
Charles Xavier      |      32 | male    | no       |
David Willimas      |      33 | male    | no       |
Edward Ulster       |      33 | male    | yes      |
Frank Townsend      |      35 | male    | no       |
Greg Sutter         |      40 | male    | no       |
Harry Rogers        |      37 | male    | no       |
John Quincy         |      43 | male    | yes      |
Kent Peterson       |      31 | male    | no       |
```

```
Larry Newell       |     23 | male    | no      |
Max Milton         |     22 | male    | no      |
Norman Lamagna     |     30 | male    | no      |
Ollie Kensinton    |     44 | male    | no      |
John Frank         |     24 | male    | no      |
Mary Williams      |     33 | female  | no      |
Jane Frank         |     38 | female  | yes     |
Jo Peterson        |     32 | female  | no      |
Angie Frost        |     33 | female  | no      |
Betty Bates        |     33 | female  | no      |
Connie Date        |     35 | female  | no      |
Donna Finster      |     20 | female  | no      |
Esther Gates       |     37 | female  | no      |
Fanny Hill         |     33 | female  | yes     |
Geta Iota          |     27 | female  | no      |
Hillary Johnson    |     22 | female  | no      |
Ingrid Kent        |     21 | female  | no      |
Jill Larson        |     20 | female  | no      |
Kim Mulrooney      |     41 | female  | no      |
Lisa Nevins        |     21 | female  | no      |
```

7.4. Отображение иерархии

Задача

Требуется вывести текст с отступами или аннотациями, чтобы показать иерархическую структуру исходного XML-документа.

Решение

Самый очевидный способ представить иерархию – сформировать с помощью пробелов отступы различной ширины. В примерах 7.29 и 7.30 приведена обобщенная таблица стилей, в которой приняты разумные допущения о преобразовании информации, хранящейся во входном документе, в иерархически организованный текст.

Пример 7.29. text.hierarchy.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings">

<xsl:include href="../strings/str.dup.xslt"/>
<xsl:include href="../strings/str.replace.xslt"/>

<xsl:output method="text"/>

<!-- По умолчанию ширина отступа на каждом уровне — два пробела -->
```

```
<xsl:param name="indent" select=" '  ' "/>

<xsl:template match="*">
    <xsl:param  name="level" select="count(./ancestor::*)"/>

    <!-- Сделать отступ для этого элемента -->
    <xsl:call-template name="str:dup" >
       <xsl:with-param name="input" select="$indent"/>
       <xsl:with-param name="count" select="$level"/>
    </xsl:call-template>

    <!-- Обработать имя элемента. По умолчанию выводится локальное имя -->
    <xsl:apply-templates select="." mode="name">
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>

    <!-- Начинаем обрабатывать атрибуты. По умолчанию выводится '(' -->
    <xsl:apply-templates select="." mode="begin-attributes">
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>

    <!-- Обработать атрибуты. По умолчанию выводятся в формате
    имя="значение". -->
    <xsl:apply-templates select="@*">
       <xsl:with-param name="element" select="."/>
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>

    <!-- Заканчиваем обработку атрибутов. По умолчанию выводится ')' -->
    <xsl:apply-templates select="." mode="end-attributes">
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>

    <!-- Обработать значение элемента. -->
    <!-- По умолчанию значение листового элемента выводится в новой
    строке с отступом -->
    <xsl:apply-templates select="." mode="value">
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>
    <xsl:apply-templates select="." mode="line-break">
       <xsl:with-param name="level" select="$level"/>
    </xsl:apply-templates>

    <!-- Обработать дочерние элементы -->
```

```
    <xsl:apply-templates select="*">
       <xsl:with-param name="level" select="$level + 1"/>
    </xsl:apply-templates>

</xsl:template>

<!-- Обработка имен элементов по умолчанию. -->
<xsl:template match="*"    mode="name">[<xsl:value-of
                                 select="local-name(.)"/></xsl:template>

<!-- Обработка начала атрибутов по умолчанию. -->
<xsl:template match="*" mode="begin-attributes">
    <xsl:if test="@*"><xsl:text> </xsl:text></xsl:if>
</xsl:template>

<!-- Обработка атрибутов по умолчанию. -->
<xsl:template match="@*">
    <xsl:value-of select="local-name(.)"/>="<xsl:value-of
                                          select="."/>"<xsl:text/>
    <xsl:if test="position() != last()">
           <xsl:text> </xsl:text>
    </xsl:if>
</xsl:template>

<!-- Обработка конца атрибутов по умолчанию. -->
<xsl:template match="*" mode="end-attributes">]</xsl:template>

<!-- Обработка значений элементов по умолчанию. -->
<xsl:template match="*" mode="value">
    <xsl:param name="level"/>

    <!-- Выводим только значения листовых элементов -->
    <xsl:if test="not(*)">
       <xsl:variable name="indent-str">
          <xsl:call-template name="str:dup" >
           <xsl:with-param name="input" select="$indent"/>
           <xsl:with-param name="count" select="$level"/>
          </xsl:call-template>
       </xsl:variable>

       <xsl:text>&#xa;</xsl:text>

       <xsl:value-of select="$indent-str"/>

       <xsl:call-template name="str:replace">
```

```
            <xsl:with-param name="input" select="."/>
            <xsl:with-param name="search-string" select=" '&#xa;' "/>
            <xsl:with-param name="replace-string"
                 select="concat('&#xa;',$indent-str)"/>
        </xsl:call-template>
    </xsl:if>
</xsl:template>

<xsl:template match="*" mode="line-break">
    <xsl:text>&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.30. Результат обработки файла ExpenseReport.xml

```
[ExpenseReport statementNum="123"]
  [Employee]
    [Name]
    Salvatore Mangano
    [SSN]
    999-99-9999
    [Dept]
    XSLT Hacking
    [EmpNo]
    1
    [Position]
    Cook
    [Manager]
    Big Boss O'Reilly
  [PayPeriod]
    [From]
    1/1/02
    [To]
    1/31/02
  [Expenses]
    [Expense]
      [Date]
      12/20/01
      [Account]
      12345
      [Desc]
      Goofing off instead of going to confrence.
      [Lodging]
      500.00
```

```
    [Transport]
    50.00
    [Fuel]
    0
    [Meals]
    300.00
    [Phone]
    100
    [Entertainment]
    1000.00
    [Other]
    300.00
  [Expense]
    [Date]
    12/20/01
    [Account]
    12345
    [Desc]
    On the beach
    [Lodging]
    500.00
    [Transport]
    50.00
    [Fuel]
    0
    [Meals]
    200.00
    [Phone]
    20
    [Entertainment]
    300.00
    [Other]
    100.00
```

XSLT 2.0

Если используется XSLT 2.0, то показанный выше код можно улучшить, воспользовавшись встроенной в XPath 2.0 функцией `replace` и показанной в рецепте 7.3 функцией `dup`.

Обсуждение

Можно возразить против конкретных параметров, которые выбраны в этой таблице стилей для вывода исходного документа в иерархическом виде. Что ж, таблица специально проектировалась так, чтобы ее можно было легко настроить. Например, вам, возможно, больше понравятся настройки, показанные в примерах 7.31 и 7.32.

Пример 7.31. Модификация таблицы стилей для вывода отчета о расходах

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

<xsl:import href="text.hierarchy.xslt"/>

<!-- Игнорировать атрибуты -->
<xsl:template match="@*"/>
<xsl:template match="*" mode="begin-attributes"/>
<xsl:template match="*" mode="end-attributes"/>

<xsl:template match="*" mode="name">
    <!--  Выводить локальное имя элемента -->
    <xsl:value-of select="local-name(.)"/>
    <!-- И, если он листовый, то еще двоеточие и пробел -->
    <xsl:if test="not(*)">: </xsl:if>
</xsl:template>

<xsl:template match="*" mode="value">
    <xsl:if test="not(*)">
       <xsl:value-of select="."/>
    </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.32. Результат применения модифицированной таблицы стилей

```
ExpenseReport
  Employee
    Name: Salvatore Mangano
    SSN: 999-99-9999
    Dept: XSLT Hacking
    EmpNo: 1
    Position: Cook
    Manager: Big Boss O'Reilly
  PayPeriod
    From: 1/1/02
    To: 1/31/02
  Expenses
    Expense
      Date: 12/20/01
      Account: 12345
      Desc: Goofing off instead of going to confrence.
      Lodging: 500.00
```

```
      Transport: 50.00
      Fuel: 0
      Meals: 300.00
      Phone: 100
      Entertainment: 1000.00
      Other: 300.00
    Expense
      Date: 12/20/01
      Account: 12345
      Desc: On the beach
      Lodging: 500.00
      Transport: 50.00
      Fuel: 0
      Meals: 200.00
      Phone: 20
      Entertainment: 300.00
      Other: 100.00
```

А, быть может, вам придется по душе формат, показанный в примерах 7.33 и 7.34, на который меня натолкнула Джени Теннисон.

Пример 7.33. tree-control.xslt

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

<xsl:import href="text.hierarchy.xslt"/>

<!-- Игнорировать атрибуты -->
<xsl:template match="@*"/>
<xsl:template match="*" mode="begin-attributes"/>
<xsl:template match="*" mode="end-attributes"/>

<xsl:template match="*"      mode="name">
    <!-- Выводить локальное имя элемента -->
    <xsl:text>[</xsl:text>
    <xsl:value-of select="local-name(.)"/>
    <!-- И, если он листовый, то еще двоеточие и пробел -->
    <xsl:text>] </xsl:text>
</xsl:template>

<xsl:template match="*" mode="value">
    <xsl:if test="not(*)">
        <xsl:value-of select="."/>
    </xsl:if>
</xsl:template>
```

```
<xsl:template match="*" mode="indent">
    <xsl:for-each select="ancestor::*">
        <xsl:choose>
           <xsl:when test="following-sibling::*"> | </xsl:when>
           <xsl:otherwise><xsl:text>   </xsl:text></xsl:otherwise>
        </xsl:choose>
    </xsl:for-each>
    <xsl:choose>
        <xsl:when test="*"> o-</xsl:when>
        <xsl:when test="following-sibling::*"> +-</xsl:when>
        <xsl:otherwise> '-</xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="*" mode="line-break">
    <xsl:text>&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.34. Форматирование результата в виде дерева

```
o-[ExpenseReport]
    o-[Employee]
    |  +-[Name] Salvatore Mangano
    |  +-[SSN] 999-99-9999
    |  +-[Dept] XSLT Hacking
    |  +-[EmpNo] 1
    |  +-[Position] Cook
    |  '-[Manager] Big Boss O'Reilly
    o-[PayPeriod]
    |  +-[From] 1/1/02
    |  '-[To] 1/31/02
    o-[Expenses]
      o-[Expense]
      |  +-[Date] 12/20/01
      |  +-[Account] 12345
      |  +-[Desc] Goofing off instead of going to confrence.
      |  +-[Lodging] 500.00
      |  +-[Transport] 50.00
      |  +-[Fuel] 0
      |  +-[Meals] 300.00
      |  +-[Phone] 100
      |  +-[Entertainment] 1000.00
```

```
|  '-[Other] 300.00
o-[Expense]
   +-[Date] 12/20/01
   +-[Account] 12345
   +-[Desc] On the beach
   +-[Lodging] 500.00
   +-[Transport] 50.00
   +-[Fuel] 0
   +-[Meals] 200.00
   +-[Phone] 20
   +-[Entertainment] 300.00
   '-[Other] 100.00
```

Можно развить эту идею и создать таблицу стилей, которая будет импортировать *tree-control.xslt* и принимать глобальный параметр, содержащий список имен элементов, для которых ветви дерева следует сворачивать. Свернутые ветви будут обозначаться префиксом x. См. примеры 7.35 и 7.36.

Пример 7.35. Таблица стилей, создающая дерево со свернутыми ветвями

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

<xsl:import href="tree-control.xslt"/>

<xsl:param name="collapse"/>
<xsl:variable name="collapse-test" select="concat(' ',$collapse,' ')"/>

<xsl:template match="*"     mode="name">
    <xsl:if test="not(ancestor::*[contains($collapse-test,
                          concat(' ',local-name(.),' '))])">
           <xsl:apply-imports/>
    </xsl:if>
</xsl:template>

<xsl:template match="*" mode="value">
    <xsl:if test="not(ancestor::*[contains($collapse-test,
                          concat(' ',local-name(.),' '))])">
           <xsl:apply-imports/>
    </xsl:if>
</xsl:template>

<xsl:template match="*" mode="line-break">
    <xsl:if test="not(ancestor::*[contains($collapse-test,
                          concat(' ',local-name(.),' '))])">
           <xsl:apply-imports/>
```

```
    </xsl:if>
</xsl:template>

<xsl:template match="*" mode="indent">
    <xsl:choose>
           <xsl:when test="self::*[contains($collapse-test,
                        concat(' ',local-name(.),' '))]">
                  <xsl:for-each select="ancestor::*">
                        <xsl:text>   </xsl:text>
                  </xsl:for-each>
                  <xsl:text> x-</xsl:text>
           </xsl:when>
           <xsl:when test="ancestor::*[contains($collapse-test,
                        concat(' ',local-name(.),' '))]"/>
           <xsl:otherwise>
                  <xsl:apply-imports/>
           </xsl:otherwise>
    </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 7.35. Результат вывода, когда $collapse="Employee PayPeriod"

```
o-[ExpenseReport]
  x-[Employee]
  x-[PayPeriod]
  o-[Expenses]
    o-[Expense]
    |  +-[Date] 12/20/01
    |  +-[Account] 12345
    |  +-[Desc] Goofing off instead of going to confrence.
    |  +-[Lodging] 500.00
    |  +-[Transport] 50.00
    |  +-[Fuel] 0
    |  +-[Meals] 300.00
    |  +-[Phone] 100
    |  +-[Entertainment] 1000.00
    |  '-[Other] 300.00
    o-[Expense]
      +-[Date] 12/20/01
      +-[Account] 12345
      +-[Desc] On the beach
      +-[Lodging] 500.00
```

```
+-[Transport] 50.00
+-[Fuel] 0
+-[Meals] 200.00
+-[Phone] 20
+-[Entertainment] 300.00
'-[Other] 100.00
```

В общем, нет конца разнообразию древовидных представлений, которые можно создать путем переопределения базовой таблицы стилей. На объектно-ориентированном жаргоне этот паттерн называется *Template Method*. Смысл его в том, что в базовом классе создается скелет алгоритма, а в подклассах переопределяются некоторые его шаги, оставляя общую структуру алгоритма неизменной. В случае XSLT роль подклассов отводится импортирующим таблицам стилей. Приводя этот пример, я не хотел убедить вас, что вывод дерева – трудное дело, вовсе нет. Истинная красота этого кода в том, что он позволяет повторно использовать одну и ту же конструкцию, уделяя внимание лишь тем аспектам, которые вы хотите изменить.

7.5. Вывод текста с нумерацией

Задача

Требуется создать текст, содержащий последовательно пронумерованные фрагменты.

Решение

Так как нумеровать можно по-разному, я в этом примере приведу ряд постепенно усложняющихся примеров, которые покрывают большую часть обычных (и ряд не столь обычных) потребностей в нумерации.

Нумерация братьев по порядку

Это самый простой вид нумерации. В примерах 7.37 и 7.38 показано, как вывести нумерованный список фамилий. Здесь я воспользовался элементом `xsl:number` и его атрибутом `count`. Атрибут `count` можно и опустить, так как по умолчанию нумеруются все узлы в текущем контексте.

Пример 7.37. Таблица стилей

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

    <xsl:template match="person">
           <xsl:number count="*" format="1. "/>
           <xsl:value-of select="@name"/>
    </xsl:template>

</xsl:stylesheet>
```

Пример 7.28. Результат

```
1. Al Zehtooney
2. Brad York
3. Charles Xavier
4. David Willimas
5. Edward Ulster
6. Frank Townsend
7. Greg Sutter
8. Harry Rogers
9. John Quincy
10. Kent Peterson
...
```

Если нужно выровнять числа по правому краю, воспользуйтесь шаблоном из рецепта 7.3.

Начало нумерации не с 1

Стандартный элемент `xsl:number` позволяет начинать нумерацию только с 1 с увеличением тоже на 1, но небольшое ухищрение позволяет это легко исправить, см. примеры 7.39 и 7.40.

Пример 7.39. Таблица стилей для нумерации не по порядку

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="person">
    <xsl:variable name="num">
       <xsl:number count="*"/>
    </xsl:variable>
    <xsl:number value="($num - 1) * 5 + 10" format="1. "/>
       <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Пример 7.40. Результат

```
10. Al Zehtooney
15. Brad York
20. Charles Xavier
25. David Willimas
30. Edward Ulster
```

```
35. Frank Townsend
40. Greg Sutter
45. Harry Rogers
50. John Quincy
55. Kent Peterson
...
```

Так можно поступать даже, если для нумерации применяются не числа. Скажем, в примерах 7.41 и 7.42 нумерация начинается с буквы L.

Пример 7.41. Таблица стилей для нумерации с буквы L

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="person">
    <xsl:variable name="num">
        <xsl:number count="*"/>
    </xsl:variable>
    <xsl:number value="$num + 11" format="A. "/>
    <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Пример 7.42. Список фамилий, пронумерованных, начиная с буквы L

```
L. Al Zehtooney
M. Brad York
N. Charles Xavier
O. David Willimas
P. Edward Ulster
Q. Frank Townsend
R. Greg Sutter
S. Harry Rogers
T. John Quincy
U. Kent Peterson
...
```

Глобальная нумерация элементов

Иногда нужно пронумеровать элементы по порядку, не обращая внимания на контекст. Типичный пример – нумерация концевых сносок. Сноски могут присутствовать на любом уровне в структуре документа, но нумероваться все равно

должны последовательно. Впрочем, продолжая уже начатую тему, рассмотрим документ, в котором люди распределены по группам и подгруппам:

```
<people>
  <group>
    <person name="Al Zehtooney" age="33" sex="m" smoker="no"/>
    <person name="Brad York" age="38" sex="m" smoker="yes"/>
    <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
    <person name="David Willimas" age="33" sex="m" smoker="no"/>
    <person name="Edward Ulster" age="33" sex="m" smoker="yes"/>
    <person name="Frank Townsend" age="35" sex="m" smoker="no"/>
  </group>
  <group>
    <person name="Greg Sutter" age="40" sex="m" smoker="no"/>
    <person name="Harry Rogers" age="37" sex="m" smoker="no"/>
    <group>
      <person name="John Quincy" age="43" sex="m" smoker="yes"/>
      <person name="Kent Peterson" age="31" sex="m" smoker="no"/>
      <person name="Larry Newell" age="23" sex="m" smoker="no"/>
      <group>
        <person name="Max Milton" age="22" sex="m" smoker="no"/>
        <person name="Norman Lamagna" age="30" sex="m" smoker="no"/>
        <person name="Ollie Kensinton" age="44" sex="m" smoker="no"/>
      </group>
      <person name="John Frank" age="24" sex="m" smoker="no"/>
    </group>
    <group>
      <person name="Mary Williams" age="33" sex="f" smoker="no"/>
      <person name="Jane Frank" age="38" sex="f" smoker="yes"/>
      <person name="Jo Peterson" age="32" sex="f" smoker="no"/>
      <person name="Angie Frost" age="33" sex="f" smoker="no"/>
      <person name="Betty Bates" age="33" sex="f" smoker="no"/>
      <person name="Connie Date" age="35" sex="f" smoker="no"/>
      <person name="Donna Finster" age="20" sex="f" smoker="no"/>
    </group>
    <group>
      <person name="Esther Gates" age="37" sex="f" smoker="no"/>
      <person name="Fanny Hill" age="33" sex="f" smoker="yes"/>
      <person name="Geta Iota" age="27" sex="f" smoker="no"/>
      <person name="Hillary Johnson" age="22" sex="f" smoker="no"/>
      <person name="Ingrid Kent" age="21" sex="f" smoker="no"/>
      <person name="Jill Larson" age="20" sex="f" smoker="no"/>
      <person name="Kim Mulrooney" age="41" sex="f" smoker="no"/>
      <person name="Lisa Nevins" age="21" sex="f" smoker="no"/>
```

```
    </group>
  </group>
</people>
```

Чтобы решить задачу, нужно лишь изменить атрибут `level` элемента `xsl:number`, присвоив ему значение `"any"`. В этом случае процессор XSLT будет при определении порядкового номера просматривать все вхождения элемента `person`. См. примеры 7.43 и 7.44.

Пример 7.43. Таблица стилей, в которой `level="any"`

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform">
    <xsl:output method="text"/>
    <xsl:strip-space elements="*"/>

    <xsl:template match="person">
           <xsl:number count="person" level="any" format="1. "/>
           <xsl:value-of select="@name"/>
        <xsl:text>&#xa;</xsl:text>
    </xsl:template>

</xsl:stylesheet>
```

Пример 7.44. Результат в случае, когда `level="any"`

```
1. Al Zehtooney
2. Brad York
3. Charles Xavier
4. David Willimas
5. Edward Ulster
6. Frank Townsend
7. Greg Sutter
8. Harry Rogers
9. John Quincy
10. Kent Peterson
11. Larry Newell
12. Max Milton
13. Norman Lamagna
14. Ollie Kensinton
15. John Frank
16. Mary Williams
17. Jane Frank
18. Jo Peterson
19. Angie Frost
20. Betty Bates
```

```
21. Connie Date
22. Donna Finster
23. Esther Gates
24. Fanny Hill
25. Geta Iota
26. Hillary Johnson
27. Ingrid Kent
28. Jill Larson
29. Kim Mulrooney
30. Lisa Nevins
```

Глобальная нумерация элементов внутри подконтекста

Бывает, что нужно ограничить глобальную нумерацию конкретным контекстом. Например, сопоставить каждому человеку номер внутри группы верхнего уровня, в которую он входит, игнорируя подгруппы.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="people/group">
    <xsl:text>Группа </xsl:text>
        <xsl:number count="group"/>
    <xsl:text>&#xa;</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

  <xsl:template match="person">
    <xsl:number count="person" level="any" from="people/group" format="1. "/>
    <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

```
Группа 1
1. Al Zehtooney
2. Brad York
3. Charles Xavier
4. David Willimas
5. Edward Ulster
6. Frank Townsend

Группа 2
1. Greg Sutter
```

```
2. Harry Rogers
3. John Quincy
4. Kent Peterson
5. Larry Newell
6. Max Milton
7. Norman Lamagna
8. Ollie Kensinton
9. John Frank
10. Mary Williams
11. Jane Frank
12. Jo Peterson
13. Angie Frost
14. Betty Bates
15. Connie Date
16. Donna Finster
17. Esther Gates
18. Fanny Hill
19. Geta Iota
20. Hillary Johnson
21. Ingrid Kent
22. Jill Larson
23. Kim Mulrooney
24. Lisa Nevins
```

Иерархическая нумерация

В формальных и юридических документах нумерация часто производится с учетом не только порядка следования, но и уровня в иерархии. В примере 7.45 демонстрируется использование элемента `xsl:number` с атрибутом `level="multiple"`.

Пример 7.45. Иерархическая нумерация людей с учетом группы

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="people/group">
    <xsl:text>Группа </xsl:text>
    <xsl:number count="group"/>
    <xsl:text>&#xa;</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

  <xsl:template match="person">
    <xsl:number count="group | person" level="multiple" format="1.1.1 "/>
```

```
    <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Нумерация, реализованная в примере 7.45, выглядит странновато, но отражает результат применения атрибута `count`, когда `level = "multiple"`. Атрибут `count` просто определяет, какие элементы-предки включать при формировании иерархического номера. Таблица стилей присвоила людям номера в зависимости от группы и позиции элемента в этой группе. Так, Bard York получил номер 1.2, поскольку он занимает вторую позицию в группе 1. Аналогично, Max Milton идет под номером 2.3.4.1, так как он находится в группе 2, если учитывать только группы верхнего уровня, в группе 3 – если учитывать группы верхнего и второго уровня, и в группе 4 – если учитывать группы верхнего, второго и третьего уровня. И, наконец, в своей собственной группе он первый:

```
Группа 1
1.1 Al Zehtooney
1.2 Brad York
1.3 Charles Xavier
1.4 David Willimas
1.5 Edward Ulster
1.6 Frank Townsend

Группа 2
2.1 Greg Sutter
2.2 Harry Rogers
2.3.1 John Quincy
2.3.2 Kent Peterson
2.3.3 Larry Newell
2.3.4.1 Max Milton
2.3.4.2 Norman Lamagna
2.3.4.3 Ollie Kensinton
2.3.5 John Frank
2.4.1 Mary Williams
2.4.2 Jane Frank
2.4.3 Jo Peterson
2.4.4 Angie Frost
2.4.5 Betty Bates
2.4.6 Connie Date
2.4.7 Donna Finster
2.5.1 Esther Gates
2.5.2 Fanny Hill
```

```
2.5.3 Geta Iota
2.5.4 Hillary Johnson
2.5.5 Ingrid Kent
2.5.6 Jill Larson
2.5.7 Kim Mulrooney
2.5.8 Lisa Nevins
```

В типичном приложении ожидается такая схема нумерации, при котором номер на любом уровне соотносится с номером на предшествующем уровне. Этого можно достичь, применив несколько элементов `xsl:number` подряд, как показано в примерах 7.46 и 7.47.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="group">
    <xsl:text>Группа </xsl:text>
    <xsl:number count="group" level="multiple"/>
    <xsl:text>&#xa;</xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="person">
    <xsl:number count="group" level="multiple" format="1.1.1."/>
    <xsl:number count="person" level="single" format="1 "/>
    <xsl:value-of select="@name"/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Пример 7.47. Результат

```
Группа 1
1.1 Al Zehtooney
1.2 Brad York
1.3 Charles Xavier
1.4 David Willimas
1.5 Edward Ulster
1.6 Frank Townsend
Группа 2
2.1 Greg Sutter
2.2 Harry Rogers
Группа 2.1
```

```
2.1.1 John Quincy
2.1.2 Kent Peterson
2.1.3 Larry Newell
Группа 2.1.1
2.1.1.1 Max Milton
2.1.1.2 Norman Lamagna
2.1.1.3 Ollie Kensinton
2.1.4 John Frank
Группа 2.2
2.2.1 Mary Williams
2.2.2 Jane Frank
2.2.3 Jo Peterson
2.2.4 Angie Frost
2.2.5 Betty Bates
2.2.6 Connie Date
2.2.7 Donna Finster
Группа 2.3
2.3.1 Esther Gates
2.3.2 Fanny Hill
2.3.3 Geta Iota
2.3.4 Hillary Johnson
2.3.5 Ingrid Kent
2.3.6 Jill Larson
2.3.7 Kim Mulrooney
2.3.8 Lisa Nevins
```

Обсуждение

С помощью одного или нескольких элементов `xsl:number` с подходящими атрибутами можно реализовать почти любую схему нумерации. Однако чрезмерное употребление `xsl:number` (особенно с атрибутом `level="multiple"`) может замедлить выполнение таблицы стилей. Для очень глубоких иерархий можно резко увеличить производительность, если передавать номер родителя дочернему элементу в качестве параметра. Посмотрите, как таким способом можно реализовать иерархическую нумерацию, вовсе не используя элемент `xsl:number`:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="group">
    <xsl:param name="parent-level" select=" '' "/>

    <xsl:variable name="number" select="concat($parent-level,position())"/>
```

```
    <xsl:text>Группа </xsl:text>
    <xsl:value-of select="$number"/>
    <xsl:text>&#xa;</xsl:text>

    <xsl:apply-templates>
       <xsl:with-param name="parent-level" select="concat($number,'.')"/>
    </xsl:apply-templates>

   </xsl:template>

   <xsl:template match="person">
    <xsl:param name="parent-level" select=" '' "/>

    <xsl:variable name="number">
    <xsl:value-of select="concat($parent-level,position(),' ')"/>
   </xsl:variable>

   <xsl:value-of select="$number"/>
   <xsl:value-of select="@name"/>
   <xsl:text>&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Такое использование функции `position()` менее удобно, если нужна нумерация с помощью букв или римских цифр.

7.6. Вывод текста в области заданной ширины с заданным выравниванием

Задача

Требуется вывести многострочный текст, присутствующий в XML-документе, в области фиксированной ширины. При этом перенос строк должен производиться по границам слов.

Решение

Вот решение, в котором перенос и выравнивание достигаются за счет повторного использования шаблона `text:justify` из рецепта 7.3. Для пущей гибкости ширину области выравнивания можно задавать независимо от ширины области переноса, считая, что по умолчанию они совпадают.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" id="text.wrap"
    xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
```

```
    xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text"
    exclude-result-prefixes="text">

<xsl:include href="../strings/str.find-last.xslt"/>
<xsl:include href="text.justify.xslt"/>

<xsl:template match="node() | @*" mode="text:wrap" name="text:wrap">
    <xsl:param name="input" select="normalize-space()"/>
    <xsl:param name="width" select="70"/>
    <xsl:param name="align-width" select="$width"/>
    <xsl:param name="align" select=" 'left' "/>

    <xsl:if test="$input">
       <xsl:variable name="line">
           <xsl:choose>
              <xsl:when test="string-length($input) > $width">
                   <xsl:call-template name="str:substring-before-last">
                      <xsl:with-param name="input"
                           select="substring($input,1,$width)"/>
                      <xsl:with-param name="substr" select=" ' ' "/>
                   </xsl:call-template>
           </xsl:when>
           <xsl:otherwise>
                  <xsl:value-of select="$input"/>
           </xsl:otherwise>
       </xsl:choose>
    </xsl:variable>

       <xsl:if test="$line">
       <xsl:call-template name="text:justify">
           <xsl:with-param name="value" select="$line"/>
              <xsl:with-param name="width" select="$align-width"/>
              <xsl:with-param name="align" select="$align"/>
           </xsl:call-template>
           <xsl:text>&#xa;</xsl:text>
       </xsl:if>

        <xsl:call-template name="text:wrap">
           <xsl:with-param name="input"
              select="substring($input, string-length($line) + 2)"/>
           <xsl:with-param name="width" select="$width"/>
           <xsl:with-param name="align-width" select="$align-width"/>
           <xsl:with-param name="align" select="$align"/>
       </xsl:call-template>
```

```
    </xsl:if>

</xsl:template>
```

В этом решении используется шаблон `str:substring-before-last` из рецепта 2.4. Основная идея заключается в том, чтобы извлечь строку длиной не более `$width` символов или меньше, если в конце окажется не пробел. Остаток строки далее обрабатывает рекурсивно. Нетривиальная часть – сделать так, чтобы слово, содержащее не менее `$width` символов, разбивалось на части.

Ниже показан результат применения этого рецепта к конкретному тексту. Для демонстрации ширина выравнивания и ширина переноса выбраны различными:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:text="http://www.ora.com/XSLTCookbook/namespaces/text">

<xsl:include href="text.wrap.xslt"/>

<xsl:strip-space elements="*"/>
<xsl:output method="text"/>

<xsl:template match="p">
    <xsl:apply-templates select="." mode="text:wrap">
           <xsl:with-param name="width" select="40"/>
           <xsl:with-param name="align" select=" 'center' "/>
           <xsl:with-param name="align-width" select="60"/>
    </xsl:apply-templates>
    <xsl:text>&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>

Входной документ:
<doc>
  <p>In the age of the internet, formats such HTML, XHTML and PDF
clearly dominate the application of XSL and XSLT. However, plain old
text will never become obsolete because it is the lowest common
denominator in both human and machine-readable formats.  XML is often
converted to text to be imported into another application that does not
know how to read XML or does not interpret it the way you would prefer.
Text output is also used when the result will be sent to a terminal or
post processed in a Unix pipeline.</p>
  <p>Many recipes in this section place stress on XSLT techniques that
create very generic XML to text converters. Here generic means that the
transformation can easily be customized to work on many different XML
inputs or produce a variety of outputs or both. The techniques employed
```

```
in these recipes have application beyond specifics of a given recipe and
often beyond the domain of text processing. In particular, you may want to
look at recipes 7.2 through 7.5 even if they do not address a present need.
</p>
</doc>
```

Результат:

```
            In the age of the internet, formats
              such HTML, XHTML and PDF clearly
             dominate the application of XSL and
              XSLT. However, plain old text will
           never become obsolete because it is the
           lowest common denominator in both human
             and machine-readable formats. XML is
            often converted to text to be imported
            into another application that does not
               know how to read XML or does not
            interpret it the way you would prefer.
              Text output is also used when the
             result will be sent to a terminal or
              post processed in a Unix pipeline.

              Many recipes in this section place
            stress on XSLT techniques that create
             very generic XML to text converters.
                 Here generic means that the
           transformation can easily be customized
           to work on many different XML inputs or
            produce a variety of outputs or both.
               The techniques employed in these
               recipes have application beyond
            specifics of a given recipe and often
            beyond the domain of text processing.
            In particular, you may want to look at
           recipes 7.2 through 7.5 even if they do
                 not address a present need.
```

Обсуждение

Во многих задачах преобразования текста выходное устройство не поддерживает строки произвольной длины. Например, терминал переносит строки, которые шире экрана. В результате вывод получается некрасивым. В этом рецепте показано, как можно выполнять форматирование в области фиксированной ширины более осмысленно.

См. также

Похожий шаблон для переноса текста приведен в книге Джени Теннисон *XSLT and XPath on the Edge (M&T, 2001)*. Однако в моем решении добавлена возможность выравнивания и корректно обрабатывается случай, когда встречаются слова длиннее заданного размера.

Глава 8. Преобразование XML в XML

Изменить и изменить к лучшему – разные вещи.
Немецкая пословица

8.0. Введение

Одно из достоинств XML заключается в том, что если какой-то документ вам не нравится, его можно изменить. Поскольку всем понравиться невозможно, то задача преобразования XML в XML возникает очень часто. Однако преобразовывать XML-документы приходится не только для того, чтобы исправить плохо продуманную структуру. Иногда нужно объединить несколько разнородных документов в один. А иногда разбить большой документ на несколько частей. Бывает так, что из документа необходимо выделить интересующую информацию, не меняя его структуры, а затем отправить результат на дальнейшую обработку.

Простой, но весьма важный инструмент, встречающийся во многих преобразованиях XML в XML, – это *тождественное преобразование*. Так называется таблица стилей, которая копирует входной документ в выходной без изменения. Возможно, вам кажется, что эта задача больше подходит для утилиты копирования операционной системы, но, как будет ясно из следующих примеров, такую таблицу стилей можно импортировать в другие таблицы, чтобы реализовать распространенные преобразования без дополнительных усилий.

В примере 1 приведена тождественная таблица стилей. Лично мне больше нравится называть ее копирующей таблицей, а технику ее использования я называю *идиомой переопределяемого копирования*.

Пример 8.1. copy.xslt

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

<xsl:template match="node() | @*">
    <xsl:copy>
           <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
</xsl:template>

</xsl:template>
```

8.1. Преобразование атрибутов в элементы

Задача

Имеется документ, в котором информация закодирована в виде атрибутов, а нужно закодировать ее в элементах.

Решение

Эта задача будто специально предназначена для применения *идиомы переопределяемого копирования*, с которой мы начали эту главу. В примере ниже все атрибуты преобразуются в элементы.

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="@*">
  <xsl:element name="{local-name(.)}" namespace="{namespace-uri(..)}">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

В этой таблице переопределяется поведение при копировании атрибутов, для чего введен шаблон, преобразующий каждый атрибут в элемент с тем же именем, содержимое которого совпадает со значением атрибута. Предполагается также, что новый элемент будет принадлежать тому же пространству имен, что и родительский узел атрибута. Если такие предположения вам не подходят, воспользуйтесь следующим кодом:

```
<xsl:template match="@*">
  <xsl:variable name="namespace">
   <xsl:choose>
    <!-- Взять пространство имен атрибута, если оно задано -->
    <xsl:when test="namespace-uri()">
      <xsl:value-of select="namespace-uri()" />
    </xsl:when>
    <!-- В противном случае взять пространство имен родителя -->
    <xsl:otherwise>
      <xsl:value-of select="namespace-uri(..)" />
    </xsl:otherwise>
   </xsl:choose>
  </xsl:variable>
```

```
  <xsl:element name="{name()}" namespace="{$namespace}">
    <xsl:value-of select="." />
  </xsl:element>
</xsl:template>
```

Часто при преобразовании атрибутов в элементы необходимо проявлять большую избирательность (см. примеры 8.2 – 8.4).

Пример 8.2. Входные данные

```
<people which="MeAndMyFriends">
    <person firstname="Sal" lastname="Mangano" age="38" height="5.75"/>
    <person firstname="Mike" lastname="Palmieri" age="28" height="5.10"/>
    <person firstname="Vito" lastname="Palmieri" age="38" height="6.0"/>
    <person firstname="Vinny" lastname="Mari" age="37" height="5.8"/>
</people>
```

Пример 8.3. Таблица стилей для преобразования только элементов person

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="person/@*">
  <xsl:element name="{local-name(.)}" namespace="{namespace-uri(..)}">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Пример 8.4. Результат

```
<people which="MeAndMyFriends">

  <person>
    <firstname>Sal</firstname>
    <lastname>Mangano</lastname>
    <age>38</age>
    <height>5.75</height>
  </person>

  <person>
```

```
    <firstname>Mike</firstname>
    <lastname>Palmieri</lastname>
    <age>28</age>
    <height>5.10</height>
  </person>

  <person>
    <firstname>Vito</firstname>
    <lastname>Palmieri</lastname>
    <age>38</age>
    <height>6.0</height>
  </person>

  <person>
    <firstname>Vinny</firstname>
    <lastname>Mari</lastname>
    <age>37</age>
    <height>5.8</height>
  </person>

</people>
```

XSLT 2.0

В XSLT 2.0 решение можно сделать более компактным, но смысл рецепта не изменяется. Ниже мы заменили громоздкую конструкцию `xsl:choose` выражением if языка XPath 2.0.

```
<xsl:template match="@*">
  <xsl:variable name="namespace"
                select="if (namespace-uri()) then namespace-uri()
                                                 else namespace-uri(..)"/>
    <xsl:element name="{name()}" namespace="{$namespace}">
       <xsl:value-of select="." />
    </xsl:element>
</xsl:template>
```

Обсуждение

В этом разделе и в рецепте 8.2 рассматриваются задачи, возникающие из-за того, что автор документа принял неудачное решение закодировать информацию в атрибутах вместо элементов. Проблема выбора между этими вариантами – один из самых противоречивых аспектов проектирования структуры документов[1]. Эти примеры полезны, так как позволяют исправить то, что вы считаете чужой ошибкой.

[1] Я знаю еще лишь один относящийся к стилю вопрос, пробуждающий в программистах еще более страстные эмоции: где ставить фигурные скобки в C-подобных языках (например, C++ и Java).

8.2. Преобразование элементов в атрибуты

Задача

Имеется документ, в котором информация представлена в виде дочерних элементов, а хотелось бы закодировать ее в атрибутах.

Решение

Можно, как и в рецепте 8.1, воспользоваться идиомой переопределяемого копирования. Однако, преобразуя элементы в атрибуты, необходимо четко определить, что именно должно преобразовываться. Ведь преобразовывать вообще все элементы просто не имеет смысла. В следующей таблице стилей обращается преобразование атрибутов в элементы, описанное в рецепте 8.1.

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8"/>

<xsl:template match="person">
  <xsl:copy>
    <xsl:for-each select="*">
      <xsl:attribute name="{local-name(.)}">
         <xsl:value-of select="."/>
      </xsl:attribute>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

XSLT 1.0

Преобразование элементов в атрибуты не всегда так прямолинейно, как противоположное. Если преобразуемые элементы сами обладают атрибутами, то нужно решить, что с ними делать. В предыдущем решении они будут просто потеряны. Альтернатива – передать их новому родителю:

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8"/>

<xsl:template match="person">
```

```
  <xsl:copy>
    <xsl:for-each select="*">
      <xsl:attribute name="{local-name(.)}">
           <xsl:value-of select="."/>
        </xsl:attribute>
      <xsl:copy-of select="@*"/>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>


</xsl:stylesheet>
```

Однако это годится лишь в случае, когда имена всех атрибутов уникальны. Если это не так, то придется переименовывать атрибуты, например, как в следующем примере:

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8"/>

<xsl:template match="person">
  <xsl:copy>
    <xsl:for-each select="*">
    <xsl:attribute name="{local-name(.)}">
       <xsl:value-of select="."/>
    </xsl:attribute>
    <xsl:variable name="elem-name" select="local-name(.)"/>
       <xsl:for-each select="@*">
          <xsl:attribute name="{concat($elem-name,'-',local-name(.))}">
             <xsl:value-of select="."/>
          </xsl:attribute>
       </xsl:for-each>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>


</xsl:stylesheet>
```

Еще одна сложность возникает, когда у элементов-братьев неуникальные имена, поскольку в этом случае при преобразовании их в атрибуты произойдет конфликт имен. Можно применить другую стратегию – создавать из элемента атрибут только в том случае, когда у элемента нет атрибутов или дочерних элементов, и у родительского элемента нет других дочерних элементов с таким же именем, и все родители не имеют атрибутов:

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" indent="yes" version="1.0" encoding="UTF-8"/>

<!-- Отобрать элементы, которые являются родителями -->
<xsl:template match="*[*]">
  <xsl:choose>
    <!-- Преобразовывать дочерние элементы, только если у этого элемента -->
    <!-- нет атрибутов -->
    <xsl:when test="not(@*)">
       <xsl:copy>
       <!-- Преобразовывать дочерний элемент в атрибут, если  -->
       <!-- у него нет ни потомков, ни атрибутов, а имена всех -->
       <!-- братьев уникальны  -->
       <xsl:for-each select="*">
          <xsl:choose>
          <xsl:when test="not(*) and not(@*) and
                          not(preceding-sibling::*[name() =
                                                   name(current())])
                         and
                          not(following-sibling::*[name() =
                                                   name(current())])">
              <xsl:attribute name="{local-name(.)}">
                  <xsl:value-of select="."/>
              </xsl:attribute>
            </xsl:when>
            <xsl:otherwise>
              <xsl:apply-templates select="."/>
            </xsl:otherwise>
            </xsl:choose>
          </xsl:for-each>
       </xsl:copy>
    </xsl:when>
    <xsl:otherwise>
       <xsl:copy>
           <xsl:apply-templates/>
       </xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0

В примере ниже мы несколько упростили и ускорили решения для XSLT 1.0, воспользовавшись командой `xsl:for-each-group`. Хитрость в том, что мы задали атрибут `group-by="name( )"` для выяснения того, есть ли братья с одинаковыми именами. Дополнительно в этом решении атрибуты преобразуемого элемента передаются родителю, что можно было бы сделать и для версии 1.0:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" indent="yes" version="1.0" encoding="UTF-8"/>

<!-- Отобрать элементы, которые являются родителями -->
<xsl:template match="*[*]">
  <xsl:choose>
  <!-- Преобразовывать дочерние элементы, только если у этого элемента -->
  <!--нет атрибутов -->
   <xsl:when test="not(@*)">
    <xsl:copy>
       <!-- Преобразовывать дочерний элемент в атрибут, если  -->
       <!-- у него нет ни потомков, ни атрибутов, а имена всех -->
       <!-- братьев уникальны  -->
       <xsl:for-each-group select="*" group-by="name()">
            <xsl:choose>
               <xsl:when test="not(*) and count(current-group()) eq 1">
                    <xsl:attribute name="{local-name(.)}">
                           <xsl:value-of select="."/>
                    </xsl:attribute>
                    <!-- Копировать атрибуты дочернего элемента
                    в родительский -->
                    <xsl:copy-of select="@*"/>
               </xsl:when>
               <xsl:otherwise>
                    <xsl:apply-templates select="current-group()"/>
               </xsl:otherwise>
            </xsl:choose>
       </xsl:for-each-group>
    </xsl:copy>
   </xsl:when>
   <xsl:otherwise>
    <xsl:copy>
       <xsl:apply-templates select="@*| node()"/>
    </xsl:copy>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Обратите внимание, что элемент `test` теоретически можно было бы преобразовать в атрибут `E1`. Собственно, наша таблица стилей даже попытается это сделать. Но потерпит неудачу, так как в XSLT копирование атрибутов допускается только перед копированием каких-либо других узлов. Если вам когда-нибудь придется иметь дело с такими неудачными документами, как этот, то надо будет выполнить два прохода. На первом проходе не преобразуйте элементы в атрибуты, а просто скопируйте их, пометив специальным атрибутом, показывающим, что они подходят для преобразования. А на втором проходе сначала преобразуйте все помеченные элементы в атрибуты их родителей, а затем скопируйте прочие дочерние элементы без изменения.

В таблицах стилей для обеих версий есть одно ограничение: предполагается, что на некотором уровне все элементы, пригодные для преобразования в атрибуты, предшествуют тем, которые для этого не годятся. Ниже приведен пример документа, не удовлетворяющего этому условию:

```
<E1>
    <E2>
        <e31>a</e31>
        <e32>b</e32>
        <e33>c</e33>
    </E2>
        <test>a</test>
    <E2>
        <e31>u</e31>
        <e32>v</e32>
        <e33>w</e33>
    </E2>
    <E2>
        <e31>x</e31>
        <e32>y</e32>
        <e33>z</e33>
    </E2>
</E1>
```

8.3. Переименование элементов или атрибутов

Задача

Требуется изменить локальные имена или имена пространств имен элементов или атрибутов в XML-документе.

Решение

Если нужно переименовать лишь небольшое количество элементов или атрибутов, воспользуйтесь прямолинейным вариантом идиомы переопределяемого копирования, как показано в примере 8.5.

Пример 8.5. Переименовать элемент person в individual

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8"/>

<xsl:template match="person">
    <individual>
           <xsl:apply-templates select="@* | node()"/>
    </individual>
</xsl:template>

</xsl:stylesheet>
```

Или по-другому, с использованием `xsl:element`:

```
...
<xsl:template match="person">
    <xsl:element name="individual">
           <xsl:apply-templates/>
    </xsl:element>
</xsl:template>
...
```

Переименовать атрибуты столь же просто:

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8"/>

<xsl:template match="@lastname">
    <xsl:attribute name="surname">
           <xsl:value-of select="."/>
    </xsl:attribute>
</xsl:template>

</xsl:stylesheet>
```

Иногда нужно изменить имя пространства имен, а не локальное имя. См. пример 8.6.

Пример 8.6. В этом документе используется пространство имен foo

```
<foo:someElement xmlns:foo="http://www.ora.com/XMLCookbook/namespaces/foo">
    <foo:aChild>
            <foo:aGrandChild/>
            <foo:aGrandChild>
            </foo:aGrandChild>
    </foo:aChild>
</foo:someElement>
```

Для каждого элемента в пространстве имен `foo` создадим новый элемент в пространстве имен `bar`, как показано в примерах 8.7 и 8.8.

Пример 8.7. Таблица стилей для замены foo на bar

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:foo="http://www.ora.com/XMLCookbook/namespaces/foo"
    xmlns:bar="http://www.ora.com/XMLCookbook/namespaces/bar">

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:strip-space elements="*"/>

<xsl:template match="foo:*">
    <xsl:element name="bar:{local-name()}">
            <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Пример 8.8. Результат

```
<bar:someElement xmlns:bar="http://www.ora.com/XMLCookbook/namespaces/bar">
    <bar:aChild>
            <bar:aGrandChild/>
            <bar:aGrandChild/>
    </bar:aChild>
</bar:someElement>
```

Обсуждение

Придумывание имен – это важное умение, которым владеют немногие практикующие программисты (ваш покорный слуга к их числу не принадлежит)[1]. Поэтому нужно знать, как изменить имя, оказавшееся неудачным.

Если нужно переименовать сразу много элементов или атрибутов, то имеет смысл прибегнуть к табличному решению, как показано в примерах 8.9 – 8.11.

Пример 8.9. Обобщенная таблица стилей для переименования, управляемая таблицей данных

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ren="http://www.ora.com/namespaces/rename">

<xsl:import href="copy.xslt"/>

<!-- Переопределить в импортирующей таблице  -->
<xsl:variable name="lookup"  select="/.."/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="*">
  <xsl:choose>
    <xsl:when test="$lookup/ren:element[@from=name(current())]">
      <xsl:element
            name="{$lookup/ren:element[@from=local-name(current())]/@to}">
         <xsl:apply-templates select="@*"/>
         <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-imports/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="@*">
  <xsl:choose>
    <xsl:when test="$lookup/ren:attribute[@from=name(current())]">
      <xsl:attribute name="{$lookup/ren:attribute[@from=name(current())]/@to}">
           <xsl:value-of select="."/>
```

[1] Свидетельством моей бездарности в области именования может служить тот факт, что мой сын прожил в этом мире два дня без всякого имени. Мы с женой просто не могли придумать ничего, что понравилось бы нам обоим. Но, понимая всю важность хорошего имени, остановились на Леонардо.

```
            </xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
       <xsl:apply-imports/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Пример 8.10. Применение показанной выше таблицы стилей

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ren="http://www.ora.com/namespaces/rename">

<xsl:import href="TableDrivenRename.xslt"/>

<!-- Загружаем справочную таблицу. Мы определили ее локально, но можно
было взять из внешнего файла -->
<xsl:variable name="lookup"  select="document('')/*[ren:*]"/>

<!-- Определяем правила переменования -->
<ren:element from="person" to="individual"/>
<ren:attribute from="firstname" to="givenname"/>
<ren:attribute from="lastname" to="surname"/>
<ren:attribute from="age" to="yearsOld"/>

</xsl:stylesheet>
```

Пример 8.11. Результат

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

<people which="MeAndMyFriends">

  <individual givenname="Sal" surname="Mangano" yearsOld="38" height="5.75"/>

  <individual givenname="Mike" surname="Palmieri" yearsOld="28" height="5.10"/>

  <individual givenname="Vito" surname="Palmieri" yearsOld="38" height="6.0"/>

  <individual givenname="Vinny" surname="Mari" yearsOld="37" height="5.8"/>

</people>
```

Этот подход применим и тогда, когда некоторые элементы или атрибуты нужно обрабатывать с учетом контекста. Рассмотрим, например, такой фрагмент документа:

```
<clubs>
  <club name="The 500 Club">
    <members>
      <member name="Joe Smith">
         <position name="president"/>
      </member>
      <member name="Jill McFonald">
         <position name="treasurer"/>
      </member>
      <!-- ... -->
    <members>
  </club>
  <!-- ... -->
<clubs>
```

Предположим, что требуется переименовать атрибут `@name` в `@title`, но только для элементов `position`. Если воспользоваться табличным решением, то атрибут `@name` будет изменен во всех элементах. Чтобы выйти из положения, создадим шаблон, который переопределяет поведение по умолчанию для всех элементов, кроме `position`.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ren="http://www.ora.com/namespaces/rename">

<xsl:import href="TableDrivenRename.xslt"/>

<!-- Загружаем справочную таблицу. Мы определили ее локально, но можно
было взять из внешнего файла -->
<xsl:variable name="lookup"  select="document('')/*[ren:*]"/>

<!-- Определяем правила переменования -->
<ren:attribute from="name" to="title"/>

<!-- ПЕРЕОПРЕДЕЛЕНИЕ: все атрибуты, не принадлежащие элементу position,
просто копируются -->
<xsl:template match="@name[not(parent::position)]">
     <xsl:copy/>
</xsl:template>

</xsl:stylesheet>
```

Если для переименования пространства имен применяется копирование, то старое пространство имен может упрямо отказываться исчезать, хотя оно больше и не нужно. Снова рассмотрим документ `foo` с дополнительным элементом из пространства имен `doc`:

```
<foo:someElement xmlns:foo="http://www.ora.com/XMLCookbook/namespaces/foo"
  xmlns:doc="http://www.ora.com/XMLCookbook/namespaces/doc">
  <foo:aChild>
    <foo:aGrandChild/>
    <foo:aGrandChild>
       <doc:doc>Эту документацию не следует ни удалять, ни изменять.
       </doc:doc>
    </foo:aGrandChild>
  </foo:aChild>
</foo:someElement>
```

Если применить к этому документу таблицу стилей для переименования пространств имен, то к элементу `doc` окажется присоединено пространство имен `foo`.

```
<bar:someElement xmlns:bar="http://www.ora.com/XMLCookbook/namespaces/bar">
  <bar:aChild>
    <bar:aGrandChild/>
    <bar:aGrandChild>
      <doc:doc xmlns:doc="http://www.ora.com/XMLCookbook/namespaces/doc"
        xmlns:foo="http://www.ora.com/XMLCookbook/namespaces/foo">
          Эту документацию не следует ни удалять, ни изменять.
      </doc:doc>
    </bar:aGrandChild>
  </bar:aChild>
</bar:someElement>
```

Объясняется это тем, что элемент `doc` обрабатывается командой `xsl:copy`. И `xsl:copy`, и `xsl:copy-of` всегда копируют все пространства имен, ассоциированные с элементом. В XSLT 2.0 у обеих этих команд есть необязательный атрибут `copy-namespaces`, который может принимать значения `yes` или `no`. Поскольку элемент `doc` является потомком элементов из пространства имен `foo`, то у него есть узел пространства имен `foo`, пусть даже во входном документе его не видно. Во избежание копирования нежелательного пространства имен воспользуемся командой `xsl:element`, чтобы не копировать элементы, а создать их заново.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:foo="http://www.ora.com/XMLCookbook/namespaces/foo"
  xmlns:bar="http://www.ora.com/XMLCookbook/namespaces/bar">

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

```
<xsl:strip-space elements="*"/>

<!-- Для каждого элемента создаем новый элемент с таким же локальным
именем и пространством имен. -->
<xsl:template match="*">
    <xsl:element   name="{name()}"   namespace="{namespace-uri()}">
       <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

<xsl:template match="foo:*">
    <xsl:element name="bar:{local-name()}">
       <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Этот прием можно применить и для удаления всех пространств имен из документа:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:strip-space elements="*"/>

<xsl:template match="*">
    <xsl:element name="{local-name()}">
       <xsl:apply-templates/>
    </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

8.4. Объединение документов с одной и той же схемой

Задача

Есть несколько документов с одинаковой структурой, а нужно объединить их в один.

Решение

Если все документы имеют разное содержимое или вы готовы смириться с дубликатами, то решение несложно:

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

<xsl:output method="xml" indent="yes"/>

<xsl:param name="doc2"/>

<xsl:template match="/*">
    <xsl:copy>
       <xsl:copy-of select="* | document($doc2)/*/*"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Если во входных документах есть дубликаты, а вы хотите, чтобы в выходном документе их не было, то для удаления дубликатов можно применить рецепт 5.1. Рассмотрим документы, представленные в примерах 8.12 и 8.13.

Пример 8.12. Документ 1

```
<people which="MeAndMyFriends">
  <person firstname="Sal" lastname="Mangano" age="38" height="5.75"/>
  <person firstname="Mike" lastname="Palmieri" age="28" height="5.10"/>
  <person firstname="Vito" lastname="Palmieri" age="38" height="6.0"/>
  <person firstname="Vinny" lastname="Mari" age="37" height="5.8"/>
</people>
```

Пример 8.13. Документ 2

```
<people which="MeAndMyCoWorkers">
  <person firstname="Sal" lastname="Mangano" age="38" height="5.75"/>
  <person firstname="Al" lastname="Zehtooney" age="33" height="5.3"/>
  <person firstname="Brad" lastname="York" age="38" height="6.0"/>
  <person firstname="Charles" lastname="Xavier" age="32" height="5.8"/>
</people>
```

Следующая таблица стилей объединяет оба документа и устраняет повторяющийся элемент с помощью команды `xsl:sort` и расширения `exsl:node-set`:

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

  <xsl:import href="exsl.xsl" />

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

```
<xsl:param name="doc2"/>
<!-- Атрибут 'key' введен для того, чтобы упростить удаление дубликатов -->
<xsl:variable name="all">
  <xsl:for-each select="/*/person | document($doc2)/*/person">
    <xsl:sort select="concat(@lastname,@firstname)"/>
    <person key="{concat(@lastname, @firstname)}">
        <xsl:copy-of select="@* | node()" />
    </person>
  </xsl:for-each>
</xsl:variable>

<xsl:template match="/">

<people>
  <xsl:for-each
    select="exsl:node-set($all)/person[not(@key =
        preceding-sibling::person[1]/@key)]">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</people>

</xsl:template>
```

У такого способа удаления дубликатов есть три недостатка. Во-первых, изменяется порядок элементов, а это может быть нежелательно. Во-вторых, требуется функция расширения node-set, которой нет в стандартном XSLT 1.0. В-третьих, эта таблица стилей не является достаточно общей, поскольку ее нужно переписывать целиком для каждого случая удаления дубликатов.

Решить эти проблемы можно с помощью элемента `xsl:key`:

```
<!-- Stylesheet: merge-simple-using-key.xslt -->
<!-- Импортировать эту таблицу стилей в другую, где определен ключ -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:merge="http:www.ora.com/XSLTCookbook/mnamespaces/merge">

<xsl:param name="doc2"/>

<xsl:template match="/*">
    <!-- Копировать самый внешний элемент исходного документа -->
    <xsl:copy>
       <!-- Для каждого дочернего элемента в исходном документе решаем,
       нужно ли копировать его в выходной, основываясь на том, есть он
       в другом документе или нет -->
       <xsl:for-each select="*">
```

```
        <!-- Вызываем шаблон, который определяет уникальный ключ для
        этого элемента. Он должен быть определен во включающей таблице
        стилей. -->
            <xsl:variable name="key-value">
                   <xsl:call-template name="merge:key-value"/>
            </xsl:variable>

            <xsl:variable name="element" select="."/>
            <!-- Этот цикл for-each нужен просто для
            переключения контекста на второй документ. -->
            <xsl:for-each select="document($doc2)/*">
               <!-- Применяем механизм ключей для проверки
               присутствия элемента во втором документе.
               Ключ должен быть определен во включающей
               таблице стилей. -->
               <xsl:if test="not(key('merge:key', $key-value))">
                   <xsl:copy-of select="$element"/>
               </xsl:if>
            </xsl:for-each>

        </xsl:for-each>

        <!-- Копируем все элементы из второго документа. -->
        <xsl:copy-of select="document($doc2)/*/*"/>

    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Следующая таблица стилей импортирует предыдущую, в ней определен ключ и шаблон для получения значения ключа:

```
<!-- Эта таблица стилей определяет уникальность элементов в терминах
ключей. -->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:merge="http:www.ora.com/XSLTCookbook/mnamespaces/merge">

<xsl:include href="merge-simple-using-key.xslt"/>

<!-- Человек уникально определяется конкатенацией имени и фамилии -->
<xsl:key name="merge:key" match="person"
           use="concat(@lastname,@firstname)"/>

<xsl:output method="xml" indent="yes"/>
```

```
<!-- Этот шаблон возвращает значение ключа элемента -->
<xsl:template name="merge:key-value">
    <xsl:value-of select="concat(@lastname,@firstname)"/>
</xsl:template>

</xsl:stylesheet>
```

Другой способ объединить документы с устранением дубликатов основан на теоретико-множественных операциях над значениями, которые обсуждаются в рецепте 9.2. Решение мы приводим здесь, но отсылаем читателя к упомянутому рецепту за дополнительной информацией. Нужные таблицы стилей приведены в примерах 8.14 и 8.15.

Пример 8.14. Повторно используемая таблица стилей, реализующая объединение документов в терминах теоретико-множественной операции объединения

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset">

<xsl:import href="../query/vset.ops.xslt"/>

<xsl:output method="xml" indent="yes"/>

<xsl:param name="doc2"/>

<xsl:template match="/*">
  <xsl:copy>
    <xsl:call-template name="vset:union">
      <xsl:with-param name="nodes1" select="*"/>
      <xsl:with-param name="nodes2" select="document($doc2)/*/*"/>
    </xsl:call-template>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Пример 8.15. Таблица стилей, определяющая, что означает равенство элементов

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset">

<xsl:import href="merge-using-vset-union.xslt"/>

<xsl:template match="person" mode="vset:element-equality">
```

```
  <xsl:param name="other"/>
  <xsl:if test="concat(@lastname,@firstname) =
                concat($other/@lastname,$other/@firstname)">
    <xsl:value-of select="true()"/>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Решение на основе шаблона `vset:union` требует меньше нового кода, чем основанное на ключах, однако для больших документов решение на базе `xsl:key`, скорее всего, окажется быстрее.

Обсуждение

В объединении документов возникает нужда в случае, когда различные части документа готовятся разными людьми или процессами. Необходимо это и тогда, когда требуется восстановить очень большой документ, который был разбит на части либо для параллельной обработки, либо потому что обрабатывать его целиком было очень неудобно.

В примерах из этого раздела рассмотрен простой случай, когда объединяются только два документа. Если нужно объединить произвольное число документов, то необходим механизм передачи списка документов в таблицу стилей. Можно, например, передать параметр, содержащий все имена файлов, перечисленные через пробел, а затем вычленить отдельные имена, воспользовавшись простым шаблоном для выделения лексем (рецепт 2.9). Другой способ заключается в передаче имен всех файлов в отдельном документе, как показано в примерах 8.16 и 8.17.

Пример 8.16. XML-файл, содержащий имена объединяемых документов

```
<mergeDocs>
    <doc path="people1.xml"/>
    <doc path="people2.xml"/>
    <doc path="people3.xml"/>
    <doc path="people4.xml"/>
</mergeDocs>
```

Пример 8.17. Таблица стилей для объединения документов (в предположении, что содержимое не дублируется)

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

<xsl:output method="xml" indent="yes"/>

<xsl:variable name="docs" select="/*/doc"/>
```

```
<xsl:template match="mergeDocs">
  <xsl:apply-templates select="doc[1]"/>
</xsl:template>

<!-- Сопоставляем с первым документом, чтобы создать элемент верхнего
уровня -->
<xsl:template match="doc">
  <xsl:variable name="path" select="@path"/>
  <xsl:for-each select="document($path)/*">
    <xsl:copy>
      <!-- Включаем в объединение дочерние элементы первого документа -->
      <xsl:copy-of select="@* | *"/>
      <!-- Цикл по остальным документам для объединения их дочерних
      элементов -->
         <xsl:for-each select="$docs[position() > 1]">
           <xsl:copy-of select="document(@path)/*/*"/>
         </xsl:for-each>
    </xsl:copy>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

8.5. Объединение документов с различными схемами

Задача

Имеется несколько документов с разной структурой, а требуется объединить их в один.

Решение

Процедура объединения структурно различных документов зависит от конкретного приложения. Поэтому мы не можем предложить одно общее решение. Однако мы попробуем прикинуть, в каких случаях обычно возникает такая необходимость, и приведем решение для каждого случая.

Включить документ в качестве подраздела родительского документа

Включение документа в качестве подраздела в другой документ – это простейшая интерпретация объединения такого вида. Идея решения состоит в том, чтобы воспользоваться командой `xsl:copy-of` для копирования всего документа или его части в подходящее место другого документа. В следующем примере мы вставляем два документа в документ-контейнер, предполагая, что имена элементов в контейнере определяют, какие файлы нужно вставлять.

```
<MyNoteBook>
  <friends>
  </friends>
  <coworkers>
  </coworkers>
  <projects>
    <project>Заменить mapML на XSLT с помощью Xalan C++</project>
    <project>Выяснить, в чем смысл жизни.</project>
    <project>Разобраться, куда деваются носки из сушилки.</project>
  </projects>
</MyNoteBook>

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="friends | coworkers">
    <xsl:copy>
       <xsl:variable name="file" select="concat(local-name(),'.xml')"/>
       <xsl:copy-of select="document($file)/*/*"/>
    </xsl:copy>
  </xsl:template>
...
</xsl:stylesheet>

<?xml version="1.0" encoding="UTF-8"?>
<MyNoteBook>
  <friends>
    <person firstname="Sal" lastname="Mangano" age="38" height="5.75"/>
    <person firstname="Mike" lastname="Palmieri" age="28" height="5.10"/>
    <person firstname="Vito" lastname="Palmieri" age="38" height="6.0"/>
    <person firstname="Vinny" lastname="Mari" age="37" height="5.8"/>
  </friends>
  <coworkers>
    <person firstname="Sal" lastname="Mangano" age="38" height="5.75"/>
    <person firstname="Al" lastname="Zehtooney" age="33" height="5.3"/>
    <person firstname="Brad" lastname="York" age="38" height="6.0"/>
    <person firstname="Charles" lastname="Xavier" age="32" height="5.8"/>
  </coworkers>
  <projects>
    <project>Заменить mapML на XSLT с помощью Xalan C++</project>
```

```
    <project>Выяснить, в чем смысл жизни.</project>
    <project>Разобраться, куда деваются носки из сушилки.</project>
  </projects>
</MyNoteBook>
```

Интересная вариация на эту тему – документ, в котором декларативно требуется включить другой документ в определенное место. Консорциум W3C определяет стандартный способ решения этой задачи – *XInclude* (*http://www.w3.org/TR/xinclude/*). На языке XSLT можно реализовать универсальный процессор `XInclude`, обобщив таблицу *copy.xslt*:

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

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" indent="yes"/>
<xsl:strip-space elements="*"/>

<xsl:template match="xi:include" xmlns:xi="http://www.w3.org/2001/
XInclude">
    <xsl:for-each select="document(@href)">
       <xsl:apply-templates/>
    </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Команда `xsl:for-each` всего лишь переключает контекст на включаемый документ. Следующая за ней команда `xsl:apply-templates` продолжает копировать содержимое включаемого документа.

Сплетение двух документов

Это вариант простого включения, когда объединяются дочерние элементы однотипных родителей. Рассмотрим двух биологов, которые порознь собирали информацию о различных животных. Перед тем как построить общую базу данных, они должны привести свои файлы к единой структуре.

У биолога 1 есть такой файл:

```
<animals>
  <mammals>
    <animal common="chimpanzee" species="Pan troglodytes" order="Primates"/>
    <animal common="human" species="Homo Sapien" family="Primates"/>
  </mammals>
  <reptiles>
   <animal common="boa constrictor" species="Boa constrictor"order="Squamata"/>
    <animal common="gecko" species="Gekko gecko" order="Squamata"/>
  </reptiles>
  <birds>
```

```
    <animal common="sea gull" species="Larus occidentalis"
                              order="Charadriiformes"/>
    <animal common="Black-Backed Woodpecker" species="Picoides arcticus"
    order="Piciformes"/>
  </birds>
</animals>
```

А у биолога 2 такой:

```
<animals>
  <mammals>
    <animal common="hippo" species="Hippopotamus amphibius"
    family=" Hippopotamidae"/>
    <animal common="arabian camel" species="Camelus dromedarius"
    family="Camelidae"/>
  </mammals>
  <insects>
    <animal common="Lady Bug" species="Adalia bipunctata"
    family="Coccinellidae"/>
    <animal common="Dung Bettle" species=" Onthophagus australis"
    family="Scarabaeidae"/>
  </insects>
  <amphibians>
    <animal common="Green Sea Turtle" species="Chelonia mydas"
    family="Cheloniidae"/>
    <animal common="Green Tree Frog" species=" Hyla cinerea"
    family="Hylidae "/>
  </amphibians>
</animals>
```

У этих файлов схемы похожи, но не идентичны. И там, и там есть класс млекопитающих (mammals), но его подуровни организованы по-разному. На уровне animal первый биолог включил информацию об отряде (order), а второй – о семействе (family). Следующая таблица стилей сплетает документы на уровне класса (второй уровень в структуре документа):

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

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:strip-space elements="*"/>

  <xsl:param name="doc2file"/>

  <xsl:variable name="doc2" select="document($doc2file)"/>
  <xsl:variable name="thisDocsClasses" select="/*/*"/>

<xsl:template match="/*">
```

```
    <xsl:copy>
      <!-- Объединить общие разделы в документах doc и doc2.
      Включить также уникальные разделы документа doc. -->
      <xsl:for-each select="*">
      <xsl:copy>
         <xsl:copy-of select="*"/>
         <xsl:copy-of select="$doc2/*/*[name() = name(current())]/*"/>
      </xsl:copy>
      </xsl:for-each>

     <!-- Включить уникальные разделы doc2 -->
     <xsl:for-each select="$doc2/*/*">
      <xsl:if test="not($thisDocsClasses[name() = name(current())])">
         <xsl:copy-of select="."/>
      </xsl:if>
     </xsl:for-each>
   </xsl:copy>
  </xsl:template>

  </xsl:stylesheet>
```

В результате применения этой таблицы стилей получается документ, который можно далее нормализовать вручную или пропустить через еще одну автоматизированную процедуру:

```
<animals>
  <mammals>
    <animal common="chimpanzee" species="Pan troglodytes" order="Primates"/>
    <animal common="human" species="Homo Sapien" order="Primates"/>
    <animal common="hippo" species="Hippopotamus amphibius"
    family=" Hippopotamidae"/>
    <animal common="arabian camel" species="Camelus dromedarius"
    family="Camelidae"/>
  </mammals>
  <reptiles>
    <animal common="boa constrictor" species="Boa constrictor"
    order="Squamata"/>
    <animal common="gecko" species="Gekko gecko" order="Squamata"/>
  </reptiles>
  <birds>
    <animal common="sea gull" species="Larus occidentalis"
    order="Charadriiformes"/>
    <animal common="Black-Backed Woodpecker" species="Picoides arcticus"
    order="Piciformes"/>
  </birds>
```

```
  <insects>
    <animal common="Lady Bug" species="Adalia bipunctata"
    family="Coccinellidae"/>
    <animal common="Dung Bettle" species=" Onthophagus australis"
    family="Scarabaeidae"/>
  </insects>
  <amphibians>
    <animal common="Green Sea Turtle" species="Chelonia mydas"
    family="Cheloniidae"/>
    <animal common="Green Tree Frog" species=" Hyla cinerea"
    family="Hylidae "/>
  </amphibians>
</animals>
```

Соединение элементов из двух документов для получения новых элементов

Менее тривиальный случай – это наложение одного документа на другой с возможным порождением дочерних элементов в зависимости от критериев сопоставления. Рассмотрим, к примеру, следующий способ объединения документов, содержащих различную информацию о людях:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:param name="doc2file"/>

  <xsl:variable name="doc2" select="document($doc2file)"/>

  <xsl:template match="person">
    <xsl:copy>
    <xsl:for-each select="@*">
       <xsl:element name="{local-name()}">
           <xsl:value-of select="."/>
       </xsl:element>
    </xsl:for-each>
    <xsl:variable name="matching-person"
           select="$doc2/*/person[@name=concat(current()/@firstname,' ',
                                              current()/@lastname)]"/>
    <xsl:element name="smoker">
       <xsl:value-of select="$matching-person/@smoker"/>
    </xsl:element>
    <xsl:element name="sex">
```

```
        <xsl:value-of select="$matching-person/@sex"/>
      </xsl:element>
     </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей решает две задачи. Во-первых, информация, представленная во входных документах в виде атрибутов, перекодируется в элементы, а, во-вторых, копируется информация из `$doc2`, отсутствующая в первом документе.

Обсуждение

Объединение XML-документов с разными схемами нельзя определить так же однозначно, как объединение структурно одинаковых документов. В этой главе были рассмотрены три интерпретации объединения, но возможны и более сложные варианты. Так, для объединения нескольких документов может потребоваться некая комбинация включения, сплетения и соединения. Поэтому, трудно написать на XSLT единую обобщенную утилиту, которая покрывала бы все мыслимые потребности. Тем не менее, приведенные примеры показывают, с чего начинать, когда нужно реализовать какую-нибудь хитроумную схему объединения.

См. также

Примеры в этом разделе касались объединения элементов, между которыми можно установить взаимно-однозначное соответствие. В рецепте 9.5 показано, как соединять данные в структурно разнородных XML-документах с помощью аналога запросов к базе данных. Эта техника применима и для объединения документов, между которыми существует отношение один-ко-многим.

8.6. Расщепление документов

Задача

Требуется разнести элементы из одного документа по нескольким поддокументам.

Решение

XSLT 1.0

В XSLT 1.0 приходится опираться на широко распространенное, но все же нестандартное расширение, которое позволяет выводить несколько документов[1].

[1] В XSLT 2.0 такая возможность стандартизована в виде нового элемента `xsl:result-document`. Подробности см. в главе 6.

В этом решении определяется подлежащий сериализации уровень в структуре документа и имя результирующего файла. Следующая таблица стилей разбивает файл *salesBySalesPerson.xml* из главы 4 на отдельные файлы – по одному для каждого продавца. Она работает для процессора Saxon. Этот процессор также допускает использование элемента `xsl:document`, если в элементе `stylesheet` задан атрибут `version ="1.1"`. Некоторые другие процессоры поддерживают элемент `exslt:document` из проекта EXSLT.org[1].

Если вы предпочитаете не пользоваться версией 1.1, можете воспользоваться расширением `saxon:output`.

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:include href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:strip-space elements="*"/>

<xsl:template match="salesperson">
  <xsl:variable name="outFile"
    select="concat('salesperson.',translate(@name,' ','_'),'.xml')"/>
  <!-- Нестандартный элемент xsl:document, поддерживаемый saxon! -->
  <xsl:document href="{$outFile}">
    <xsl:copy>
           <xsl:copy-of select="@*"/>
        <xsl:apply-templates/>
    </xsl:copy>
  </xsl:document>
</xsl:template>

<xsl:template match="salesBySalesperson">
  <xsl:apply-templates/>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Хотя приведенная выше таблица стилей ориентирована на Saxon, подобная техника работает в большинстве процессоров XSLT 1.0 с минимальными изменениями. В Saxon поддерживается также нестандартный элемент `saxon:output` (`xmlns:saxon ="http://icl.com/saxon"`). В процессоре Xalan есть расширение `xalan:redirect` (`xmlns:xalan = "http://xml.apache.org/xalan"`).

[1] XSLT 1.1 более не считается официальной версией. С выпуском спецификации XSLT 2.0 она объявлена недействующей.

Интересная вариация на тему расщепления – дополнительное порождение выходного файла, который включает сгенерированные подфайлы с помощью команд `xinclude`:

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:import href="copy.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
<xsl:strip-space elements="*"/>

<xsl:template match="salesperson">
  <xsl:variable name="outFile"
     select="concat('salesperson.',translate(@name,' ','_'),'.xml')"/>
  <xsl:document href="{$outFile}">
     <xsl:copy>
            <xsl:copy-of select="@*"/>
        <xsl:apply-templates/>
     </xsl:copy>
  </xsl:document>

  <xi:include href="{$outFile}"
           xmlns:xi="http://www.w3.org/2001/XInclude"/>

</xsl:template>

</xsl:stylesheet>
```

Если вас беспокоит, что ваш процессор XSLT в один прекрасный день начнет поддерживать спецификацию `XInclude` и ошибочно попытается включить тот файл, который только что был выведен, можете заменить элемент `xi:include` на `xsl:element`:

```
<xsl:element name="xi:include"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xsl:attribute name="href">
        <xsl:value-of select="$outFile"/>
    </xsl:attribute>
</xsl:element>
```

См. также

В рецепте 14.1 приводятся дополнительные примеры использования различных расширений для вывода документов.

8.7. Уплощение иерархии XML

Задача

Имеется документ, в котором иерархия элементов имеет больше уровней, чем вам хотелось бы. Требуется выполнить уплощение дерева – избавиться от некоторых уровней.

Решение

Если ваша цель – убрать лишние уровни вместе с хранящейся там информацией, то можно применить идиому переопределяемого копирования. Во включающей таблице стилей должен быть шаблон, который сопоставляется с отбрасываемыми элементами и выполняет команду `apply-templates` без копирования.

Рассмотрим следующий входной документ, в котором люди разбиты на две категории: `salaried` и `union`.

```
<people>
  <union>
    <person>
      <firstname>Warren</firstname>
      <lastname>Rosenbaum</lastname>
      <age>37</age>
      <height>5.75</height>
    </person>
    <person>
      <firstname>Dror</firstname>
      <lastname>Seagull</lastname>
      <age>28</age>
      <height>5.10</height>
    </person>
    <person>
      <firstname>Mike</firstname>
      <lastname>Heavyman</lastname>
      <age>45</age>
      <height>6.0</height>
    </person>
    <person>
      <firstname>Theresa</firstname>
      <lastname>Archul</lastname>
      <age>37</age>
      <height>5.5</height>
    </person>
  </union>
  <salaried>
```

```
    <person>
      <firstname>Sal</firstname>
      <lastname>Mangano</lastname>
      <age>37</age>
      <height>5.75</height>
    </person>
    <person>
      <firstname>Jane</firstname>
      <lastname>Smith</lastname>
      <age>28</age>
      <height>5.10</height>
    </person>
    <person>
      <firstname>Rick</firstname>
      <lastname>Winters</lastname>
      <age>45</age>
      <height>6.0</height>
    </person>
    <person>
      <firstname>James</firstname>
      <lastname>O'Riely</lastname>
      <age>33</age>
      <height>5.5</height>
    </person>
  </salaried>
</people>
```

Следующая таблица стилей просто отбрасывает лишние структурные уровни:

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

    <xsl:import href="copy.xslt"/>

    <xsl:output method="xml" version="1.0" encoding="UTF-8"/>

    <xsl:template match="people">
        <xsl:copy>
           <!-- отбрасываем предков элементов person -->
           <xsl:apply-templates select="*/person" />
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Обсуждение

Наличие дополнительных структурных уровней в документе обычно можно лишь приветствовать, так как это упрощает его обработку с помощью XSLT. Однако, если их слишком много, то документ становится громоздким, и человеку воспринять его сложно. Людям свойственно делать выводы о логической структуре документа, исходя из его пространственной организации, а не из дополнительной синтаксический разметки.

В примере ниже показано, что дополнительные структурные уровни не избыточны, а несут в себе информацию. Если в процессе уплощения нужно сохранить информацию о структуре, то имеет смысл создать для ее хранения атрибут или дочерний элемент.

В следующей таблице стилей создается атрибут:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8"
  omit-xml-declaration="yes"/>

  <!-- отбрасываем родителей элементов person -->
  <xsl:template match="*[person]">
     <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="person">
    <xsl:copy>
      <xsl:apply-templates select="@*"/>
      <xsl:attribute name="class">
           <xsl:value-of select="local-name(..)"/>
      </xsl:attribute>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

А здесь создается элемент:

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

  <xsl:import href="copy.xslt"/>

  <xsl:strip-space elements="*"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" />

  <!-- отбрасываем родителей элементов person -->
  <xsl:template match="*[person]">
```

```
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="person">
    <xsl:copy>
       <xsl:copy-of select="@*"/>
       <xsl:element name="class">
            <xsl:value-of select="local-name(..)"/>
       </xsl:element>
       <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Элемент `xsl:strip-space` и атрибут `indent="yes"` элемента `xsl:output` нужны для того, чтобы в выходном файле не было лишних пустых строк, показанных ниже:

```
<people>
...
  <person>
    <class>union</class>
    <firstname>Warren</firstname>
    <lastname>Rosenbaum</lastname>
    <age>37</age>
    <height>5.75</height>
  </person>
                  <- Дырка!
  <person>
    <class>salaried</class>
    <firstname>Sal</firstname>
    <lastname>Mangano</lastname>
    <age>37</age>
    <height>5.75</height>
  </person>
...
 </people>
```

8.8. Углубление иерархии XML

Задача

Имеется плохо спроектированный документ, которому не помешали бы дополнительные структурные уровни[1].

[1] Может, для каких-то целей он и хорошо спроектирован, но у вас цели другие.

Решение

Эта задача противоположна рассмотренной в рецепте 8.7. Здесь нужно добавить в документ структурные уровни, возможно, для того чтобы организовать элементы по какому-нибудь дополнительному критерию.

Добавление структуры на основе уже имеющихся данных

Показанное ниже преобразование обратно уплощению, выполненному в рецепте 8.7.

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="people">
    <union>
         <xsl:apply-templates select="person[@class = 'union']" />
    </union>
    <salaried>
     <xsl:apply-templates select="person[@class = 'salaried']" />
    </salaried>
  </xsl:template>

</xsl:stylesheet>
```

Добавление структуры для исправления плохо спроектированного документа

Ошибочно стремясь упростить XML-разметку, некоторые авторы пытаются закодировать информацию, вставляя элементы-братья вместо родительских элементов[1].

Предположим, например, что кто-то решил следующим образом различать работников, получающих оклад и являющихся членами профсоюза:

```
<people>
  <class name="union"/>
  <person>
    <firstname>Warren</firstname>
    <lastname>Rosenbaum</lastname>
    <age>37</age>
    <height>5.75</height>
  </person>
...
  <person>
```

[1] Честно говоря, не всегда такой подход следует считать ошибочным. Проектирование – это результат поиска компромиссов между конфликтующими целями.

```
    <firstname>Theresa</firstname>
    <lastname>Archul</lastname>
    <age>37</age>
    <height>5.5</height>
  </person>
  <class name="salaried"/>
  <person>
    <firstname>Sal</firstname>
    <lastname>Mangano</lastname>
    <age>37</age>
    <height>5.75</height>
  </person>
...
  <person>
    <firstname>James</firstname>
    <lastname>O'Riely</lastname>
    <age>33</age>
    <height>5.5</height>
  </person>
</people>
```

Обратите внимание, что элементы `class`, определяющие принадлежность той или другой категории, теперь пусты. Имелось в виду, что все последующие братья элемента `class` считаются принадлежащими указанному классу, пока не встретится другой элемент `class` или не останется братьев. Такой способ кодирования легко воспринимается на взгляд, но обрабатывать его с помощью XSLT сложнее. Чтобы исправить ситуацию, понадобится таблица стилей, которая вычисляет разность между множеством элементов `person`, следующих за первым и последующим вхождениями элемента `class`. В XSLT 1.0 нет функции для явного вычисления разности множеств. Добиться того же эффекта, причем более эффективно, можно, рассмотрев те элементы, следующие за элементом `class`, позиция которых меньше, чем у элементов, следующих за последующим элементом `class`:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <!-- Общее число людей -->
  <xsl:variable name="num-people" select="count(/*/person)"/>

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

```
    <!-- Последняя интересующая нас позиция. -->
    <xsl:variable name="pos"
            select="$num-people -
              count(following-sibling::class/following-sibling::person)"/>
    <xsl:element name="{@name}">
        <!-- Скопировать людей, следующих за этим элементом class, позиция
        которых меньше или равна $pos.-->
        <xsl:copy-of
              select="following-sibling::person[position() &lt;= $pos]"/>
      </xsl:element>
  </xsl:template>

  <!-- Игнорировать элементы person. Они уже скопированы выше. -->
  <xsl:template match="person"/>

</xsl:stylesheet>
```

Более тонкое решение получается при использовании ключа:

```
<xsl:key name="people" match="person"
           use="preceding-sibling::class[1]/@name" />

<xsl:template match="people">
    <people>
      <xsl:apply-templates select="class" />
    </people>
</xsl:template>

<xsl:template match="class">
    <xsl:element name="{@name}">
      <xsl:copy-of select="key('people', @name)" />
    </xsl:element>
</xsl:template>
```

Еще одна альтернатива – пошаговый подход:

```
<xsl:template match="people">
  <people>
    <xsl:apply-templates select="class[1]" />
  </people>
</xsl:template>

<xsl:template match="class">
  <xsl:element name="{@name}">
    <xsl:apply-templates select="following-sibling::*[1][self::person]" />
```

```
    </xsl:element>
    <xsl:apply-templates select="following-sibling::class[1]" />
  </xsl:template>

  <xsl:template match="person">
    <xsl:copy-of select="." />
    <xsl:apply-templates select="following-sibling::*[1][self::person]" />
  </xsl:template>
```

XSLT 2.0

Добавление структуры на основе уже имеющихся данных

Имеющаяся в XSLT 2.0 команда `xsl:for-each-group` позволяет написать более общее решение, чем возможно в версии 1.0. Хотя и для 1.0 существуют обобщенные решения (см. обсуждение), но все они сложнее:

```
<xsl:template match="people">
  <xsl:for-each-group select="person"
                      group-by="preceding-sibling::class[1]/@name">
    <xsl:element name="{curent-grouping-key()">
      <xsl:apply-templates select="current-group()" />
    </xsl:element>
  </xsl:for-each>
</xsl:template>
```

Добавление структуры для исправления плохо спроектированного документа

Можно воспользоваться командой `xsl:for-each-group` с атрибутом `starting-with`:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="people">
    <xsl:copy>
       <xsl:for-each-group select="*" group-starting-with="class">
      <xsl:element name="{@name}">
         <xsl:apply-templates  select="current-group()[not(self::class)]"/>
      </xsl:element>
     </xsl:for-each-group>
    </xsl:copy>
```

```
  </xsl:template>

</xsl:stylesheet>
```

Обсуждение

Добавление структуры на основе уже имеющихся данных

Добавляя структуру на основе уже имеющихся данных, мы явно ссылались на критерий выделения представляющих интерес категорий (например, `union` и `salaried`). Лучше, если бы таблица стилей умела самостоятельно находить эти категории. Тогда она стала бы более общей, пусть даже ценой усложнения:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <!-- Строим список всех уникальных классов -->
  <xsl:variable name="classes"
        select="/*/*/@class[not(. = ../preceding-sibling::*/@class)]"/>
  <xsl:template match="/*">
    <!-- Для каждого класса создаем элемент с именем, которое задано
    в элементе class. Он будет содержать все элементы, принадлежащие
    данному классу. -->
    <xsl:for-each select="$classes">
      <xsl:variable name="class-name" select="."/>
      <xsl:element name="{$class-name}">
           <xsl:for-each select="/*/*[@class=$class-name]">
             <xsl:copy>
                  <xsl:apply-templates/>
             </xsl:copy>
           </xsl:for-each>
      </xsl:element>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

Хотя эта таблица стилей и не общая на все 100%, но в ней нет предположений о том, какие классы имеются в документе. Единственная информация, специфичная для приложения, – это тот факт, что категории закодированы в атрибуте `@class`, и этот атрибут встречается в элементах, отстоящих от корня на два уровня.

Добавление структуры для исправления плохо спроектированного документа

Решение можно реализовать явно в терминах разности множеств. Хотя оно выглядит элегантно, но для больших документов, включающих много категорий, непрактично. Для вычисления разности мы воспользовались трюком, о котором рассказано в рецепте 9.1:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="class">
    <!-- Все люди, следующие за элементом class -->
    <xsl:variable name="nodes1" select="following-sibling::person"/>
    <!-- Все люди, следующие за последующим элементом class -->
    <xsl:variable name="nodes2"
           select="following-sibling::class/following-sibling::person"/>
    <xsl:element name="{@name}">
      <xsl:copy-of select="$nodes1[count(. | $nodes2) != count($nodes2)]"/>
    </xsl:element>
  </xsl:template>

  <xsl:template match="person"/>

</xsl:stylesheet>
```

8.9. Реорганизация иерархии XML

Задача

Требуется реорганизовать информацию в XML-документе, так чтобы неявно закодированная информация стала явной и наоборот.

Решение

XSLT 1.0

Снова рассмотрим документ *SalesBySalesPerson.xml* из главы 4:

```
<salesBySalesperson>
  <salesperson name="John Adams" seniority="1">
    <product sku="10000" totalSales="10000.00"/>
    <product sku="20000" totalSales="50000.00"/>
```

```
    <product sku="25000" totalSales="920000.00"/>
  </salesperson>
  <salesperson name="Wendy Long" seniority="5">
    <product sku="10000" totalSales="990000.00"/>
    <product sku="20000" totalSales="150000.00"/>
    <product sku="30000" totalSales="5500.00"/>
  </salesperson>
  <salesperson name="Willie B. Aggressive" seniority="10">
    <product sku="10000" totalSales="1110000.00"/>
    <product sku="20000" totalSales="150000.00"/>
    <product sku="25000" totalSales="2920000.00"/>
    <product sku="30000" totalSales="115500.00"/>
    <product sku="70000" totalSales="10000.00"/>
  </salesperson>
  <salesperson name="Arty Outtolunch" seniority="10"/>
</salesBySalesperson>
```

Информация о том, какие товары и на какую сумму продал каждый продавец, представлена явно. Но общая выручка от продажи каждого товара закодирована неявно, как и сведения обо всех продавцах, сумевших продать данный товар.

Для реорганизации этого документа нужно преобразовать его к виду, где будут показаны итоги по каждому товару. Это делает следующая таблица стилей:

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

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:key name="sales_key" match="salesperson" use="product/@sku"/>

<xsl:variable name="products" select="//product"/>
<xsl:variable name="unique-products"
    select="$products[not(@sku = preceding::product/@sku)]"/>

<xsl:template match="/">
  <salesByProduct>
    <xsl:for-each select="$unique-products">
     <xsl:variable name="sku" select="@sku"/>
     <xsl:copy>
       <xsl:copy-of select="$sku"/>
       <xsl:attribute name="totalSales">
           <xsl:value-of select="sum($products[@sku=$sku]/@totalSales)"/>
       </xsl:attribute>
       <xsl:for-each select="key('sales_key',$sku)">
```

```
            <xsl:copy>
               <xsl:copy-of select="@*"/>
               <xsl:attribute name="sold">
                 <xsl:value-of select="product[@sku=$sku]/@totalSales"/>
               </xsl:attribute>
            </xsl:copy>
         </xsl:for-each>
      </xsl:copy>
     </xsl:for-each>
   </salesByProduct>
</xsl:template>

</xsl:stylesheet>
```

Результирующий документ выглядит так:

```
<salesByProduct>
 <product sku="10000" totalSales="2110000">
  <salesperson name="John Adams" seniority="1" sold="10000.00"/>
  <salesperson name="Wendy Long" seniority="5" sold="990000.00"/>
  <salesperson name="Willie B. Aggressive" seniority="10" sold="1110000.00"/>
 </product>
 <product sku="20000" totalSales="350000">
  <salesperson name="John Adams" seniority="1" sold="50000.00"/>
  <salesperson name="Wendy Long" seniority="5" sold="150000.00"/>
  <salesperson name="Willie B. Aggressive" seniority="10" sold="150000.00"/>
 </product>
 <product sku="25000" totalSales="3840000">
  <salesperson name="John Adams" seniority="1" sold="920000.00"/>
  <salesperson name="Willie B. Aggressive" seniority="10" sold="2920000.00"/>
 </product>
 <product sku="30000" totalSales="121000">
  <salesperson name="Wendy Long" seniority="5" sold="5500.00"/>
  <salesperson name="Willie B. Aggressive" seniority="10" sold="115500.00"/>
 </product>
 <product sku="70000" totalSales="10000">
  <salesperson name="Willie B. Aggressive" seniority="10" sold="10000.00"/>
 </product>
</salesByProduct>
```

Альтернативное решение основано на методе Мюнха. В нем для извлечения уникальных товаров применяется элемент `xsl:key`. Выражение `$products[count(.|key('product_key', @sku)[1]) = 1]` отбирает первый товар в каждой группе, причем группировка производится по атрибуту `sku`:

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

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:variable name="doc" select="/"/>

<xsl:key name="product_key" match="product" use="@sku"/>
<xsl:key name="sales_key" match="salesperson" use="product/@sku"/>

<xsl:variable name="products" select="//product"/>

<xsl:template match="/">
  <salesByProduct>
    <xsl:for-each select="$products[count(.|key('product_key',@sku)[1])
                          = 1]">
      <xsl:variable name="sku" select="@sku"/>
      <xsl:copy>
        <xsl:copy-of select="$sku"/>
        <xsl:attribute name="totalSales">
          <xsl:value-of select="sum(key('product_key',$sku)/@totalSales)"/>
        </xsl:attribute>
        <xsl:for-each select="key('sales_key',$sku)">
         <xsl:copy>
                 <xsl:copy-of select="@*"/>
                 <xsl:attribute name="sold">
                  <xsl:value-of select="product[@sku=$sku]/@totalSales"/>
                </xsl:attribute>
             </xsl:copy>
          </xsl:for-each>
        </xsl:copy>
      </xsl:for-each>
    </salesByProduct>
</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0

В XSLT 2.0 задача решается прямолинейно за счет применения команды `xsl:for-each-group`:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/">
  <salesByProduct>
```

```
<!-- Сгруппировать товары по sku -->
<xsl:for-each-group select="//product" group-by="@sku">
   <xsl:copy>
  <xsl:copy-of select="@sku"/>
  <!-- Для подведения итогов по товару используем
  current-group() -->
  <xsl:attribute name="totalSales"
           select="format-number(sum(current-group()/@totalSales),'#')"/>
  <!-- Копируем элементы salesperson, которые содержат в качестве
  дочернего элемент со значением sku для текущей группы товаров -->
         <xsl:for-each select="/*/salesperson">
            <xsl:if test="product[@sku eq current-grouping-key()]">
                <xsl:copy>
                   <xsl:copy-of select="@*"/>
                </xsl:copy>
            </xsl:if>
         </xsl:for-each>
      </xsl:copy>
      </xsl:for-each-group>
  </salesByProduct>
 </xsl:template>
</xsl:stylesheet>
```

Обсуждение

XSLT 1.0

Это решение годится только для одного конкретного приложения. И тут ничего не поделаешь. Написать обобщенную таблицу стилей для реорганизации документа трудно, если вообще возможно, поскольку преобразование сильно зависит от природы преобразовываемого документа.

Однако можно сформулировать соображения, применимые ко многим видам реорганизации.

Во-первых, поскольку дерево документа реорганизуется полностью, маловероятно, что в основе решения будет лежать только сопоставление и применение шаблонов. Скорее, таблицы стилей будут итеративными, то есть в них будет широко использоваться команда `xsl:for-each`.

Во-вторых, почти всегда придется инициализировать глобальные переменные для хранения элементов, извлеченных с нижних уровней структуры XML-документа. Кроме того, вероятно, придется выделять подмножество, содержащее уникальные элементы такого рода. Детальное обсуждение методов построения таких подмножество см. в рецепте 5.3.

В-третьих, реорганизация часто включает агрегирование данных с вычислением сумм, произведений и более сложных агрегатов. Методы агрегирования рассматриваются в главах 3 и 16.

XSLT 2.0

Ясно, что команда `xsl:for-each-group` заметно упрощает реорганизацию XML-документов. Главное – четко уяснить, на основе какого критерия производится группировка, а затем реорганизовать элементы по их соотношению с группой.

См. также

В рецепте 6.2 приводятся дополнительные примеры использования команды `for-each-group` в XSLT 2.0

Глава 9. Опрос XML-документа

```
<xsl:template name="child-query">
<xsl:with-param name="parent" select="'Daddy'"/>
<xsl:value-of select="concat('But, why',$parent,'?')"/>
<xsl:apply-templates select="reasonable_response"/>
<xsl:call-template name="child-query">
<xsl:with-param name="parent" select="$parent"/>
</xsl:call-template> </xsl:template>
                        <!-- Родители, не распознающие
                        хвостовую рекурсию, рискуют
                        переполнить стек. -->
```

9.0. Введение

Эта глава посвящена рецептам использования XSLT как языка запросов для XML. Под запросом к XML понимается извлечение информации из одного или нескольких XML-документов, которая отвечает на вопросы о хранящихся в этих документах фактах и существующих между ними отношениях. Можно провести аналогию между запросом к XML на языке XSLT и запросом к реляционной базе данных на языке SQL.

«Официальным» языком запросом для XML является не XSLT, а XQuery (*http://www.w3.org/TR/xquery/*). У языков XSLT и XQuery много общего, но есть и очевидные различия. Например, и XSLT, и XQuery полагаются на XPath. Однако XSLT-сценарий записывается в синтаксисе XML, тогда как для программ на XQuery определены две синтаксических нотации: в виде XML и в виде, больше ориентированном на человека (*http://www.w3.org/TR/xqueryx*).

Когда была впервые выдвинута идея языка запросов для XML, отличного от XSLT, отношение к ней было противоречивым. Многие члены сообщества XML полагали, что эти языки будут слишком сильно перекрываться. И действительно, любой запрос, сформулированный на XQuery, можно реализовать также и на XSLT. Причем во многих случаях решение на XSLT оказывается таким же компактным, как и на XQuery. Преимущество языка XQuery заключается в том, что в общем случае понять написанную на нем программу проще, чем эквивалентную программу на XSLT. Для тех, кто знаком с языком SQL, освоение XQuery не представит серьезных сложностей. Конечно, легкость овладения новым материалом – это еще и функция от того, к чему вы привыкли, поэтому все сравнения не абсолютны.

Детальное рассмотрение языка XQuery и сравнение его с XSLT выходит за рамки настоящей главы. Вместо этого мы приведем примеры запросов для тех, кто уже потратил время на изучение XSLT и не хочет учить еще один относящийся к XML язык.

Невозможно предложить примеры, исчерпывающие все многообразие запросов к XML-данным. Поэтому мы поступим иначе. Сначала приведем примеры примитивных запросов с достаточно широкой областью применимости. Они станут кирпичиками, из которых будут строиться более сложные запросы. Затем мы покажем, как можно подойти к решению большинства задач, приведенных в качестве примеров в документе *XML Query Use Cases*, который опубликован на сайте W3C (*http://www.w3.org/TR/xmlquery-use-cases*). Для многих актуальных задач вы сможете найти в этом документе похожий сценарий. А затем уже не составит труда адаптировать решение к вашим XML-данным.

9.1. Выполнение теоретико-множественных операций над наборами узлов

Задача

Требуется найти объединение, пересечение, разность или симметрическую разность двух наборов узлов. Требуется также сравнить два набора узлов на равенство и входимость.

Решение

XSLT 1.0

Объединение ищется тривиально, так как язык XPath поддерживает эту операцию напрямую:

```
<xsl:copy-of select="$node-set1 | $node-set2"/>
```

Для вычисления пересечения двух наборов узлов потребуется более сложное выражение:

```
<xsl:copy-of select="$node-set1[count(. | $node-set2) = count($node-set2)]"/>
```

То, что таким образом мы находим все элементы `node-set1`, которые также принадлежат `node-set2`, следует из того факта, что объединение `node-set2` и некоторого элемента из `node-set1`, одновременно входящего в `node-set2`, не приводит к изменению `node-set2`.

Разность множеств (то есть совокупность элементов, входящих в первое множество и не входящих во второе) вычисляется следующим образом:

```
<xsl:copy-of select="$node-set1[count(. | $node-set2) != count($node-set2)]"/>
```

То, что таким образом мы находим все элементы `node-set1`, которые не принадлежат `node-set2`, следует из того факта, что объединение `node-set2`

и некоторого элемента из `node-set1`, не входящего в `node-set2`, порождает множество с большим количеством элементов.

Ниже приведен шаблон для вычисления симметрической разности (совокупность элементов, входящих ровно в одно из двух множеств):

```
<xsl:copy-of select="$node-set1[count(. | $node-set2) != count($node-set2)] |
$node-set2[count(. | $node-set1) != count($node-set1)] "/>
```

Симметрическая разность – это просто объединение двух разностей: между первым и вторым и между вторым и первым множеством.

Проверка равенства множеств `node-set1` и `node-set2`:

```
<xsl:if test="count($ns1|$ns2) = count($ns1) and
              count($ns1) = count($ns2)">
```

Два множества равны, если число элементов в их объединении равно сумме числа элементов в каждом множестве по отдельности.

Проверка того, что `node-set2` – подмножество `node-set1`:

```
<xsl:if test="count($node-set1|$node-set2) = count($node-set1)">
```

Проверка того, что `node-set2` – истинное подмножество `node-set1`:

```
<xsl:if test="count($ns1|$ns2) = count($ns1) and count($ns1) > count(ns2)">
```

XSLT 2.0

В XPath 2.0 операции над множествами встроены. Детали см. в рецепте 1.7.

Обсуждение

Может возникнуть вопрос, что общего между операциями над множествами и запросами к XML. Теоретико-множественные операции дают средства для нахождения сходства и различий между множествами элементов документа. А многие базовые запросы, предъявляемые к данным, так или иначе связаны с отысканием сходства и различий.

Рассмотрим, к примеру, такой способ извлечения элементов `person` из файла *people.xml*:

```
<xsl:variable name="males" select="//person[@sex='m']"/>
<xsl:variable name="females" select="//person[@sex='f']"/>
<xsl:variable name="smokers" select="//person[@smoker='yes']"/>
<xsl:variable name="non-smokers" select="//person[@smoker='no']"/>
```

С точки зрения страхования жизни имеет смысл назначать сумму страховки в зависимости от характеристик страхующегося:

```
<!-- Курящие мужчины -->
<xsl:variable name="super-risk"
    select="$males[count(. | $smokers) = count($smokers)]"/>
```

```
<!-- Курящие женщины -->
<xsl:variable name="high-risk"
    select="$females[count(. | $smokers) = count($smokers)]"/>
<!-- Некурящие мужчины -->
<xsl:variable name="moderate-risk"
    select="$males[count(. | $non-smokers) = count($non-smokers)]"/>
<!-- Некурящие женщины -->
<xsl:variable name="low-risk"
    select="$females[count(. | $non-smokers) = count($non-smokers)]"/>
```

Возможно, вы заметили, что те же ответы можно получить проще, применив логические операции вместо теоретико-множественных:

```
<!-- Курящие мужчины -->
<xsl:variable name="super-risk"
    select="//person[@sex='m' and @smoker='y']"/>
<!-- Курящие женщины -->
<xsl:variable name="high-risk"
    select="//person[@sex='f' and @smoker='y']"/>
<!-- Некурящие мужчины -->
<xsl:variable name="moderate-risk"
    select="//person[@sex='m' and @smoker='n']"/>
<!-- Некурящие женщины -->
<xsl:variable name="low-risk"
    select="//person[@sex='f' and @smoker='n']"/>
```

А если уже имеются множества мужчин и женщин, то еще эффективнее было бы написать так:

```
<!-- Курящие мужчины -->
<xsl:variable name="super-risk"
    select="$males[@smoker='y']"/>
<!-- Курящие женщины -->
<xsl:variable name="high-risk"
    select="$females[@smoker='y']"/>
<!-- Некурящие мужчины -->
<xsl:variable name="moderate-risk"
    select="$males[@smoker='n']"/>
<!-- Некурящие женщины -->
<xsl:variable name="low-risk"
    select="$females[@smoker='n']"/>
```

Но эти наблюдения не отменяют полезности теоретико-множественного подхода. Отметим, что операции над множествами работают, ничего не зная о том, что находится в множествах, то есть на более высоком уровне абстракции. Представьте себе, что имеется сложный XML-документ, и вам интересны следующие четыре множества:

```
<!-- Все элементы, у которых есть дочерние элементы c1 или c2 -->
<xsl:variable name="set1" select="//*[c1 or c2]"/>
<!-- Все элементы, у которых есть дочерние элементы c3 или c4 -->
<xsl:variable name="set2" select="//*[c3 and c4]"/>
<!-- Все элементы, у родителя которых есть атрибут  a1-->
<xsl:variable name="set3" select="//*[../@a1]"/>
<!-- Все элементы, у родителя которых есть атрибут  a2-->
<xsl:variable name="set4" select="//*[../@a2]"/>
```

В нашем исходном примере было очевидно, что множества мужчин и женщин не пересекаются (как и множества курящих и некурящих). Здесь же такого априорного знания нет. Множества могут не пересекаться, полностью совпадать или иметь несколько общих элементов. Есть только два способа выяснить, что общего между множествами `set1` и `set3`. Первый – вычислить их пересечение, второй – обойти весь документ еще раз, применяя логическую операцию `and` к определяющим эти множества предикатам. В данном случае очевидно, что пересечение – более правильный путь.

В проекте EXSLT определен модуль set, в который включены функции для выполнения рассмотренных выше операций над множествами. Там применяется интересная техника для возвращения результата операции. Вместо того чтобы возвращать результат напрямую, к нему применяется шаблон в режиме, соответствующем типу операции. Например, вычислив пересечение, функция `set:intersection` вызывает для результата команду `<xsl:apply-templates mode="set:intersection"/>`. В EXSLT определен шаблон по умолчанию, работающий в этом режиме; он возвращает копию результата в виде фрагмента дерева узлов. Такой способ возврата позволяет переопределить этот шаблон в импортирующей таблице стилей и тем самым подвергнуть результат дальнейшей обработке. Этот прием полезен, но имеет ограничения. Полезен он потому, что позволяет отказаться от функции расширения `node-set()`, преобразующей результат обратно в набор узлов. А ограничен, потому что для каждой операции в импортирующей таблице стилей каждый шаблон можно переопределить не более одного раза. Однако иногда результаты пересечения, выполненного в разных местах одной и той же таблицы стилей, нужно затем обрабатывать совершенно по-разному.

Не переживайте, если вам сразу не удалось ухватить суть только что рассмотренного приема, применяемого в EXSLT. В главе 16 мы обсудим и этот прием, и другие способы сделать XSLT-код повторно используемым.

См. также

Описание теоретико-множественных операций в EXSLT можно найти на странице *http://www.exslt.org/set/index.html*.

9.2. Выполнение теоретико-множественных операций над наборами узлов с использованием семантики значений

Задача

Требуется найти объединение, пересечение, разность или симметрическую разность двух наборов узлов. Однако теперь *равенство* определено не как *идентичность наборов узлов*, а как функция от значений узлов.

Решение

XSLT 1.0

Потребность в этом решении может возникнуть при работе с несколькими документами. Рассмотрим два документа с одной и той же DTD-схемой, в которых не может быть элементов с одинаковыми значениями. С точки зрения XSLT, элементы, взятые из разных документов, различаются, даже если у них одинаковые пространства имен, атрибуты и текстовые значения. См. примеры 9.1 – 9.4.

Пример 9.1. people1.xml

```
<people>
    <person name="Brad York" age="38" sex="m" smoker="yes"/>
    <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
    <person name="David Willimas" age="33" sex="m" smoker="no"/>
</people>
```

Пример 9.2. people2.xml

```
<people>
    <person name="Al Zehtooney" age="33" sex="m" smoker="no"/>
    <person name="Brad York" age="38" sex="m" smoker="yes"/>
    <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
</people>
```

Пример 9.3. Неудачная попытка воспользоваться операцией объединения в XSLT для устранения дубликатов

```
<xsl:template match="/">
  <people>
    <xsl:copy-of select="//person | document('people2.xml')//person"/>
  </people>
</xsl:template>
```

Пример 9.4. Результат, когда на вход подан файл people1.xml

```
<people>
    <person name="Brad York" age="38" sex="m" smoker="yes"/>
    <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
    <person name="David Willimas" age="33" sex="m" smoker="no"/>
    <person name="Al Zehtooney" age="33" sex="m" smoker="no"/>
    <person name="Brad York" age="38" sex="m" smoker="yes"/>
    <person name="Charles Xavier" age="32" sex="m" smoker="no"/>
</people>
```

Полагаться на идентичность узлов нельзя и в случае единственного документа, если вы хотите сравнивать узлы по значениям текста или каких-нибудь атрибутов.

Следующая таблица стилей – это повторно используемая реализация объединения, пересечения и разности множеств для случая, когда понятие равенства основано на семантике значений. Идея в том, что импортирующая таблица переопределит шаблон, для которого `mode="vset:element-equality"`. Тем самым она сможет определить семантику равенства, имеющую смысл для конкретных входных данных.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset">

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<!-- Реализация равенства элементов по умолчанию. Переопределить
в импортирующей таблице стилей, если необходимо. -->
<xsl:template match="node() | @*" mode="vset:element-equality">
  <xsl:param name="other"/>
  <xsl:if test=". = $other">
       <xsl:value-of select="true()"/>
  </xsl:if>
</xsl:template>

<!-- По умолчанию для проверки членства в множестве используется
равенство элементов. Вряд ли придется часто переопределять этот шаблон
в импортирующей таблице стилей. -->
<xsl:template match="node() | @*" mode="vset:member-of">
  <xsl:param name="elem"/>
  <xsl:variable name="member-of">
    <xsl:for-each select=".">
       <xsl:apply-templates select="." mode="vset:element-equality">
           <xsl:with-param name="other" select="$elem"/>
       </xsl:apply-templates>
    </xsl:for-each>
  </xsl:variable>
```

```
  <xsl:value-of select="string($member-of)"/>
</xsl:template>

<!-- Объединение двух множеств вычисляется с помощью равенства
"по значению". -->
<xsl:template name="vset:union">
  <xsl:param name="nodes1" select="/.." />
  <xsl:param name="nodes2" select="/.." />
  <!-- для внутреннего использования -->
  <xsl:param name="nodes" select="$nodes1 | $nodes2" />
  <xsl:param name="union" select="/.." />
  <xsl:choose>
    <xsl:when test="$nodes">
       <xsl:variable name="test">
            <xsl:apply-templates select="$union" mode="vset:member-of">
               <xsl:with-param name="elem" select="$nodes[1]" />
            </xsl:apply-templates>
       </xsl:variable>
       <xsl:call-template name="vset:union">
            <xsl:with-param name="nodes" select="$nodes[position() > 1]" />
            <xsl:with-param name="union"
                       select="$union | $nodes[1][not(string($test))]" />
       </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
       <xsl:apply-templates select="$union" mode="vset:union" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- По умолчанию возвращается копия объединения. Переопределить
в импортирующей таблице стилей, если нужно получить результат как
"обратный вызов". -->
<xsl:template match="/ | node() | @*" mode="vset:union">
  <xsl:copy-of select="."/>
</xsl:template>

<!-- Пересечение двух множеств вычисляется с помощью равенства
"по значению". -->
<xsl:template name="vset:intersection">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <!-- для внутреннего использования -->
  <xsl:param name="intersect" select="/.."/>
```

```
  <xsl:choose>
    <xsl:when test="not($nodes1)">
       <xsl:apply-templates select="$intersect" mode="vset:intersection"/>
    </xsl:when>
    <xsl:when test="not($nodes2)">
       <xsl:apply-templates select="$intersect" mode="vset:intersection"/>
    </xsl:when>
    <xsl:otherwise>
       <xsl:variable name="test1">
       <xsl:apply-templates select="$nodes2" mode="vset:member-of">
           <xsl:with-param name="elem" select="$nodes1[1]"/>
       </xsl:apply-templates>
    </xsl:variable>
    <xsl:variable name="test2">
       <xsl:apply-templates select="$intersect" mode="vset:member-of">
           <xsl:with-param name="elem" select="$nodes1[1]"/>
       </xsl:apply-templates>
    </xsl:variable>
    <xsl:choose>
       <xsl:when test="string($test1) and not(string($test2))">
           <xsl:call-template name="vset:intersection">
              <xsl:with-param name="nodes1"
                   select="$nodes1[position() > 1]"/>
              <xsl:with-param name="nodes2" select="$nodes2"/>
              <xsl:with-param name="intersect"
                   select="$intersect | $nodes1[1]"/>
           </xsl:call-template>
       </xsl:when>
       <xsl:otherwise>
           <xsl:call-template name="vset:intersection">
              <xsl:with-param name="nodes1"
                   select="$nodes1[position() > 1]"/>
              <xsl:with-param name="nodes2" select="$nodes2"/>
              <xsl:with-param name="intersect" select="$intersect"/>
           </xsl:call-template>
       </xsl:otherwise>
    </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- По умолчанию возвращается копия пересечения. Переопределить
в импортирующей таблице стилей, если нужно получить результат как
"обратный вызов". -->
```

```
<xsl:template match="/ | node() | @*" mode="vset:intersection">
  <xsl:copy-of select="."/>
</xsl:template>

<!-- Разность двух множеств (node1 - nodes2) вычисляется с помощью
равенства "по значению". -->
<xsl:template name="vset:difference">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <!-- для внутреннего использования -->
  <xsl:param name="difference" select="/.."/>

  <xsl:choose>
    <xsl:when test="not($nodes1)">
       <xsl:apply-templates select="$difference" mode="vset:difference"/>
   </xsl:when>
   <xsl:when test="not($nodes2)">
       <xsl:apply-templates select="$nodes1" mode="vset:difference"/>
   </xsl:when>
   <xsl:otherwise>
    <xsl:variable name="test1">
       <xsl:apply-templates select="$nodes2" mode="vset:member-of">
           <xsl:with-param name="elem" select="$nodes1[1]"/>
       </xsl:apply-templates>
    </xsl:variable>
    <xsl:variable name="test2">
       <xsl:apply-templates select="$difference" mode="vset:member-of">
           <xsl:with-param name="elem" select="$nodes1[1]"/>
       </xsl:apply-templates>
    </xsl:variable>
    <xsl:choose>
       <xsl:when test="string($test1) or string($test2)">
           <xsl:call-template name="vset:difference">
                  <xsl:with-param name="nodes1"
                     select="$nodes1[position() > 1]"/>
                  <xsl:with-param name="nodes2"
                     select="$nodes2"/>
                  <xsl:with-param name="difference"
                     select="$difference"/>
           </xsl:call-template>
       </xsl:when>
       <xsl:otherwise>
           <xsl:call-template name="vset:difference">
                  <xsl:with-param name="nodes1"
```

```
                    select="$nodes1[position() > 1]"/>
                <xsl:with-param name="nodes2"
                    select="$nodes2"/>
                <xsl:with-param name="difference"
                    select="$difference | $nodes1[1]"/>
            </xsl:call-template>
        </xsl:otherwise>
    </xsl:choose>
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>

<!-- По умолчанию возвращается копия разности. Переопределить
в импортирующей таблице стилей, если нужно получить результат как
"обратный вызов". -->
<xsl:template match="/ | node() | @*" mode="vset:difference">
    <xsl:copy-of select="."/>
</xsl:template>
```

Эти рекурсивные шаблоны реализованы исходя из следующих определений:

```
Union(nodes1,nodes2)
```

Объединение включает все узлы из набора `nodes2` плюс те узлы из набора `nodes1`, которые не входят в `nodes2`.

```
Intersection(nodes1,nodes2)
```

Пересечение включает все узлы из набора `nodes1`, которые входят также в набор `nodes2`.

```
Difference(nodes1,nodes2)
```

Разность включает все узлы из набора `nodes1`, которые не входят в набор `nodes2`.

Во всех случаях для установления факта вхождения в множество используется равенство строковых значений, но импортирующая таблица стилей может этот критерий переопределить.

Имея набор теоретико-множественных операций с семантикой значений, можно получить для файлов *people1.xml* и *people2.xml* требуемый результат, воспользовавшись такой таблицей стилей:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset">

<xsl:import href="set.ops.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

```
<xsl:template match="/">
  <people>
    <xsl:call-template name="vset:union">
       <xsl:with-param name="nodes1" select="//person"/>
       <xsl:with-param name="nodes2" select="document('people2.xml')//person"/>
   </xsl:call-template>
  </people>
</xsl:template>

<!-- Считаем, что два элемента person равны, если совпадают
атрибуты name -->
<xsl:template match="person" mode="vset:element-equality">
  <xsl:param name="other"/>
  <xsl:if test="@name = $other/@name">
    <xsl:value-of select="true()"/>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0

Главное усовершенствование, введенное в XSLT 2.0, – полноценные функции и последовательности. Это устраняет необходимость рекурсии и трюка с обратным вызовом, а определения получаются более элегантными и удобными для применения. При этом функции `vset:element-equality` и `vset:member-of` по-прежнему можно переопределять в импортирующей таблице стилей, добиваясь нужного поведения.

```
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset">

<!-- Реализация равенства элементов по умолчанию. Переопределить
в импортирующей таблице стилей, если необходимо. -->
<xsl:function name="vset:element-equality" as="xs:boolean">
    <xsl:param name="item1" as="item()?"/>
    <xsl:param name="item2" as="item()?"/>
    <xsl:sequence select="$item1 = $item2"/>
</xsl:function>

<!-- По умолчанию для проверки членства в множестве используется
равенство элементов. Вряд ли придется часто переопределять этот шаблон
в импортирующей таблице стилей. -->
<xsl:function name="vset:member-of" as="xs:boolean">
  <xsl:param name="set" as="item()*"/>
```

```
  <xsl:param name="elem" as="item()"/>
  <xsl:variable name="member-of" as="xs:boolean*"
      select="for $test in $set
                  return if (vset:element-equality($test, $elem))
                             then true() else ()"/>
  <xsl:sequence select="not(empty($member-of))"/>
</xsl:function>

<!-- Объединение двух множеств вычисляется с помощью равенства "по
значению". -->
<xsl:function name="vset:union" as="item()*">
  <xsl:param name="nodes1" as="item()*"  />
  <xsl:param name="nodes2" as="item()*" />
  <xsl:sequence select="$nodes1, for $test in $nodes2
                                 return if (vset:member-of($nodes1,$test))
                                        then () else $test"/>
</xsl:function>

<!-- Пересечение двух множеств вычисляется с помощью равенства "по
значению". -->
<xsl:function name="vset:intersection" as="item()*">
  <xsl:param name="nodes1" as="item()*" />
  <xsl:param name="nodes2" as="item()*" />
  <xsl:sequence select="for $test in $nodes1
                          return if (vset:member-of($nodes2,$test))
                                 then $test else ()"/>
</xsl:function>

<!-- Разность двух множеств (node1 - nodes2) вычисляется с помощью
равенства "по значению". -->

<xsl:function name="vset:difference" as="item()*">
  <xsl:param name="nodes1" as="item()*" />
  <xsl:param name="nodes2" as="item()*" />
  <xsl:sequence select="for $test in $nodes1 return if (vset:member-
      of($nodes2,$test)) then () else $test"/>
</xsl:function>

</xsl:stylesheet>
```

Обсуждение

Быть может, вам кажется, что равенство – вещь тривиальная; два предмета либо равны, либо нет. Однако в программировании (как и в политике), равенство зависит от того, кто сравнивает. В типичном документе элемент ассоциируется

с уникально идентифицируемым объектом. Например, один абзац ... отличается от любого другого абзаца в том же документе, даже если их содержимое одинаково. Поэтому теоретико-множественные операции, в основе которых лежат уникальные идентификаторы элементов, определены корректно. Но, если операции XSLT производятся над несколькими документами или над элементами, возникшими в результате применения команды `xsl:copy`, то нужно более аккуратно определять, какие элементы мы хотим считать равными.

Вот несколько примеров запросов, в которых требуется семантика значений:

1. Есть два документа из разных пространств имен. В примерах 9.5 – 9.8 мы ищем (локальные) имена элементов, общие для обоих документов и уникальные в своих пространствах имен.

Пример 9.5. doc1.xml

```
<doc xmlns:doc1="doc1" xmlns="doc1">
  <chapter label="1">
    <section label="1">
       <p>
           Давным-давно...
       </p>
    </section>
  </chapter>
  <chapter label="2">
    <note to="editor">Я все еще жду своего аванса в сумме $100000.</note>
    <section label="1">
       <p>
           ... и с тех пор они жили счастливо.
       </p>
    </section>
  </chapter>
</doc>
```

Пример 9.6. doc2.xml

```
<doc xmlns:doc1="doc2" xmlns="doc2">
  <chapter label="1">
    <section label="1">
       <sub>
           <p>
              Давным-давно...
              <ref type="footnote" label="1"/>
           </p>
       </sub>
       <fig>Figure1</fig>
    </section>
```

```
    <footnote label="1">
       Hey diddle diddle.
    </footnote>
  </chapter>
  <chapter label="2">
    <section label="1">
       <p>
           ... и с тех пор они жили счастливо.
       </p>
    </section>
  </chapter>
</doc>
```

Пример 9.7. unique-element-names.xslt

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:doc1="doc1" xmlns:doc2="doc2"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset"
  extension-element-prefixes="vset">

    <xsl:import href="set.ops.xslt"/>

    <xsl:output method="text" />

    <xsl:template match="/">
      <xsl:text>&#xa;Общие элементы: </xsl:text>
      <xsl:call-template name="vset:intersection">
         <xsl:with-param name="nodes1" select="//*"/>
         <xsl:with-param name="nodes2" select="document('doc2.xml')//*"/>
      </xsl:call-template>

      <xsl:text>&#xa;Элементы, имеющиеся только в doc1: </xsl:text>
      <xsl:call-template name="vset:difference">
         <xsl:with-param name="nodes1" select="//*"/>
         <xsl:with-param name="nodes2" select="document('doc2.xml')//*"/>
      </xsl:call-template>

      <xsl:text>&#xa;Элементы, имеющиеся только в doc2: </xsl:text>
      <xsl:call-template name="vset:difference">
         <xsl:with-param name="nodes1" select="document('doc2.xml')//*"/>
         <xsl:with-param name="nodes2" select="//*"/>
      </xsl:call-template>
      <xsl:text>&#xa;</xsl:text>

    </xsl:template>
```

```
    <xsl:template match="*" mode="vset:intersection">
      <xsl:value-of select="local-name(.)"/>
      <xsl:if test="position() != last()">
         <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:template>

    <xsl:template match="*" mode="vset:difference">
      <xsl:value-of select="local-name(.)"/>
      <xsl:if test="position() != last()">
         <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:template>

    <xsl:template match="doc1:* | doc2:*" mode="vset:element-equality">
      <xsl:param name="other"/>
      <xsl:if test="local-name(.) = local-name($other)">
         <xsl:value-of select="true()"/>
      </xsl:if>
    </xsl:template>

</xsl:stylesheet>
```

Пример 9.8. Результат

```
Общие элементы: doc, chapter, section, p
Элементы, имеющиеся только в doc1: note
Элементы, имеющиеся только в doc2: sub, ref, fig, footnote
```

2. Документ на языке Visio XML состоит из главных форм, экземпляров главных форм и определенных пользователем форм, для которых главной формы не существует. Требуется получить данные обо всех уникальных формах. С точки зрения этого запроса две формы считаются равными, если выполняется хотя бы одно из следующих условий:
   a. У обеих форм есть атрибут `@Master`, и значения этого атрибута совпадают.
   b. Хотя бы у одной формы нет атрибута `@Master`, но все геометрические элементы `Geom` равны. Два элемента `Geom` считаются равными, если равны все атрибуты всех их потомков.

   В противном случае формы не равны.

   Для реализации запроса можно взять пересечение множества всех форм с самим собой при описанной выше интерпретации равенства[1].

[1] Математик скажет, что пересечение любого множества с самим собой дает то же самое множество. Это так для настоящих множеств (без дубликатов). Здесь же мы имеем специально определенное понятие равенства, а наборы узлов как правило не являются настоящими множествами при таком определении равенства. Однако операции над множествами значений всегда порождают настоящие множества, и этот факт тоже можно использовать для удаления дубликатов.

Можно также воспользоваться шаблоном `vset:union` с параметром nodes:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:vxd="urn:schemas-microsoft-com:office:visio"
  xmlns:vset="http:/www.ora.com/XSLTCookbook/namespaces/vset"
  extension-element-prefixes="vset">

  <xsl:import href="set.ops.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="/">
    <UniqueShapes>
      <xsl:call-template name="vset:intersection">
        <xsl:with-param name="nodes1" select="//vxd:Pages/*/*/vxd:Shape"/>
        <xsl:with-param name="nodes2" select="//vxd:Pages/*/*/vxd:Shape"/>
       </xsl:call-template>
    </UniqueShapes>
  </xsl:template>

  <xsl:template match="vxd:Shape" mode="vset:intersection">
    <xsl:copy-of select="." />
  </xsl:template>

  <xsl:template match="vxd:Shape" mode="vset:element-equality">
    <xsl:param name="other"/>
    <xsl:choose>
      <xsl:when test="@Master and $other/@Master and @Master = $other/@Master">
            <xsl:value-of select="true()"/>
       </xsl:when>
       <xsl:when test="not(@Master) or not($other/@Master)">
            <xsl:variable name="geom1">
               <xsl:for-each select="vxd:Geom//*/@*">
                    <xsl:sort select="name()"/>
                    <xsl:value-of select="."/>
               </xsl:for-each>
            </xsl:variable>
            <xsl:variable name="geom2">
               <xsl:for-each select="$other/vxd:Geom//*/@*">
                    <xsl:sort select="name()"/>
                    <xsl:value-of select="."/>
               </xsl:for-each>
            </xsl:variable>
            <xsl:if test="$geom1 = $geom2">
                    <xsl:value-of select="true()"/>
            </xsl:if>
```

```
        </xsl:when>
      </xsl:choose>
</xs l:template>

</xsl:stylesheet>
```

9.3. Сравнение наборов узлов на равенство по значению

Задача

Требуется определить, равны ли (по значению) узлы из одного набора узлам из другого набора (порядок игнорируется).

Решение

Эта задача несколько сложнее, чем кажется на первый взгляд. Рассмотрим очевидное решение, которое во многих случаях будет работать:

```
<xsl:template name="vset:equal-text-values">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <xsl:choose>
   <!-- Пустые наборы узлов имеют равные значения -->
   <xsl:when test="not($nodes1) and not($nodes2)">
    <xsl:value-of select="true()"/>
   </xsl:when>
   <!-- Наборы из разного числа узлов не могут иметь равные значения -->
   <xsl:when test="count($nodes1) != count($nodes2)"/>
   <!-- Если элемент из набора nodes1 присутствует в наборе nodes2,
    то наборы узлов имеют одинаковые значения, если то же самое можно
    сказать о наборах, из которых общий элемент удален. -->
   <xsl:when test="$nodes1[1] = $nodes2">
    <xsl:call-template name="vset:equal-text-values">
           <xsl:with-param name="nodes1" select="$nodes1[position()>1]"/>
           <xsl:with-param name="nodes2"
                          select="$nodes2[not(. = $nodes1[1])]"/>
    </xsl:call-template>
   </xsl:when>
   <xsl:otherwise/>
 </xsl:choose>
</xsl:template>
```

Мы выбрали для этого шаблона имя `equal-text-values`, чтобы подчеркнуть контекст, в котором он должен применяться. Именно, равенство значений означает равенство текстовых значений. Очевидно, что этот шаблон не даст правильного

результата, если равенство определено на основе значений атрибутов или еще более сложного критерия. Однако, в нем есть и более тонкая ошибка. Молчаливо предполагается, что сравниваемые наборы узлов – настоящие множества (то есть не содержат дубликатов) относительно сравнения по текстовому значению. Иногда это неверно. Рассмотрим следующий XML-документ, описывающий читателей, взявших книги в библиотеке:

```
<?xml version="1.0" encoding="UTF-8"?
<library>
  <book>
    <name>High performance Java programming.</name>
    <borrowers>
           <borrower>James Straub</borower>
    </borrowers>
  </book>
  <book>
    <name>Exceptional C++</name>
    <borrowers>
           <borrower>Steven Levitt</borower>
    </borrowers>
  </book>
  <book>
    <name>Design Patterns</name>
    <borrowers>
           <borrower>Steven Levitt</borower>
           <borrower>James Straub</borower>
           <borrower>Steven Levitt</borower>
    </borrowers>
  </book>
  <book>
    <name>The C++ Programming Language</name>
    <borrowers>
           <borrower>James Straub</borower>
           <borrower>James Straub</borower>
           <borrower>Steven Levitt</borower>
    </borrowers>
  </book>
</library>
```

Если имя читателя появляется более одного раза, это просто означает, что он несколько раз брал книгу. Если нужно найти книги, которые брали одни и те же люди, то вы, наверное, согласитесь, что *Design Patterns* и *The C++ Programming Language* должны быть найдены. Однако, если в реализации такого запроса воспользоваться шаблоном `vset:equal-text-values`, то результат будет иным, поскольку в нем предполагается, что множества не содержат дубликатов.

Чтобы разрешить дубликаты, можно модифицировать `vset:equal-text-values` следующим образом:

```
<xsl:template name="vset:equal-text-values-ignore-dups">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <xsl:choose>
    <!-- Пустые наборы узлов имеют равные значения -->
     <xsl:when test="not($nodes1) and not($nodes2)">
       <xsl:value-of select="true()"/>
       </xsl:when>
     <!-- Если элемент из набора nodes1 присутствует в наборе nodes2,
    то наборы узлов имеют одинаковые значения, если то же самое можно
    сказать о наборах, из которых общий элемент удален. -->
    <!-- удалить эту строку
       <xsl:when test="count($nodes1) != count($nodes2)"/> -->
    <xsl:when test="$nodes1[1] = $nodes2">
       <xsl:call-template name="vset:equal-text-values">
           <xsl:with-param name="nodes1"
                           select="$nodes1[not(. = $nodes1[1])]"/>
          <xsl:with-param name="nodes2"
           select="$nodes2[not(. = $nodes1[1])]"/>      </xsl:call-template>
     </xsl:when>
     <xsl:otherwise/>
  </xsl:choose>
</xsl:template>
```

Обратите внимание, что проверка на равенство размеров закомментирована, поскольку при наличии дубликатов она была бы лишней. Например, в одном наборе может быть три вхождения элемента с текстовым значением `foo`, а в другом – только одно. Если дубликаты игнорируются, то эти наборы следует считать равными. Кроме того, на шаге рекурсии недостаточно просто удалить первый элемент; необходимо удалить все элементы с таким же значением, как у первого, как то делается для второго набора. При этом на каждом шаге рекурсии дубликаты учитываются в полной мере. После таких изменений все проверки на равенство, базирующиеся на текстовом значении, оказываются корректными, но за это приходится платить дополнительной обработкой множеств, которые заведомо не равны.

Описанные выше проверки на равенство не такие общие, как в операциях над множествами значений из рецепта 9.2, поскольку предполагается, что единственный вид равенства, который нас интересует, – это равенство текстовых значений. Можно обобщить шаблон, повторно воспользовавшись рассмотренной ранее техникой проверки на членство, основанной на равенстве элементов, которое можно переопределить в импортирующей таблице:

```
<xsl:template name="vset:equal">
  <xsl:param name="nodes1" select="/.."/>
```

```
  <xsl:param name="nodes2" select="/.."/>
  <xsl:if test="count($nodes1) = count($nodes2)">
    <xsl:call-template name="vset:equal-impl">
           <xsl:with-param name="nodes1" select="$nodes1"/>
           <xsl:with-param name="nodes2" select="$nodes2"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<!-- Зная, что число элементов в обоих наборах одинаково, -->
<!-- осталось только проверить, что каждый член первого набора -->
<!-- является и членом второго -->
<xsl:template name="vset:equal-impl">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <xsl:choose>
     <xsl:when test="not($nodes1)">
      <xsl:value-of select="true()"/>
     </xsl:when>
     <xsl:otherwise>
      <xsl:variable name="test">
         <xsl:apply-templates select="$nodes2" mode="vset:member-of">
           <xsl:with-param name="elem" select="$nodes1[1]"/>
         </xsl:apply-templates>
    </xsl:variable>
    <xsl:if test="string($test)">
       <xsl:call-template name="vset:equal-impl">
         <xsl:with-param name="nodes1" select="$nodes1[position() > 1]"/>
           <xsl:with-param name="nodes2" select="$nodes2"/>
       </xsl:call-template>
    </xsl:if>
   </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Если вы хотите, чтобы обобщенное равенство работало и при наличии дубликатов, то придется применить более грубый подход, выполнив два прохода по наборам узлов:

```
<xsl:template name="vset:equal-ignore-dups">
    <xsl:param name="nodes1" select="/.."/>
    <xsl:param name="nodes2" select="/.."/>

  <xsl:variable name="mismatch1">
   <xsl:for-each select="$nodes1">
```

```
    <xsl:variable name="test-elem">
       <xsl:apply-templates select="$nodes2" mode="vset:member-of">
           <xsl:with-param name="elem" select="."/>
       </xsl:apply-templates>
    </xsl:variable>
    <xsl:if test="not(string($test-elem))">
           <xsl:value-of select=" 'false' "/>
    </xsl:if>
   </xsl:for-each>
 </xsl:variable>
 <xsl:if test="not($mismatch1)">
   <xsl:variable name="mismatch2">
    <xsl:for-each select="$nodes2">
       <xsl:variable name="test-elem">
           <xsl:apply-templates select="$nodes1" mode="vset:member-of">
              <xsl:with-param name="elem" select="."/>
           </xsl:apply-templates>
       </xsl:variable>
       <xsl:if test="not(string($test-elem))">
           <xsl:value-of select=" 'false' "/>
       </xsl:if>
    </xsl:for-each>
   </xsl:variable>
   <xsl:if test="not($mismatch2)">
    <xsl:value-of select="true()"/>
   </xsl:if>
 </xsl:if>
</xsl:template>
```

В этом шаблоне во время обхода первого набора узлов ищутся элементы, отсутствующие во втором. Если таких элементов не найдено, то переменная `$mismatch1` будет равна null. В таком случае нужно повторить проверку, но на этот раз обойти второй набор узлов.

Обсуждение

Необходимость сравнивать множества на равенство часто возникает при выполнении запросов. Рассмотрим следующие задачи:

- найти все книги, написанные одними и теми же авторами;
- найти всех поставщиков, поставляющих одинаковый набор деталей;
- найти все семьи, в которых есть дети одного возраста.

Всякий раз, как вы встречаетесь с отношением один-ко-многим и требуется найти элементы первого множества, с которыми ассоциированы одинаковые наборы элементов второго множества, возникает необходимость в сравнении множеств на равенство.

9.4. Выполнение запросов, сохраняющих структуру

Задача

В ответ на запрос к XML-документу требуется получить ответ, структура которого идентична структуре исходного документа.

Решение

Запросы, сохраняющие структуру, отфильтровывают ненужную информацию, сохраняя структуру документа практически неизменной. Степень схожести структуры входного и выходного документа и есть метрика, определяющая применимость данного рецепта. Чем она больше, тем выше применимость.

В следующем примере есть два компонента: один можно использовать повторно, а другой зависит от конкретного приложения. Повторное использование допускает таблица стилей, которая копирует все узлы в выходной документ (тождественное преобразование). Эта таблица, показанная в примере 9.9, широко применялась в главе 6.

Пример 9.9. copy.xslt

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

<xsl:template match="node() | @*">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Другой компонент – это таблица стилей, которая импортирует *copy.xslt* и содержит правила, переопределяющие поведение по умолчанию. Так, следующая таблица стилей порождает файл, аналогичный *people.xml*, в котором оставлены только курящие женщины.

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

  <xsl:import href="copy.xslt"/>

  <!-- Схлопнуть дыры, оставшиеся после удаления элементов person -->
  <xsl:strip-space elements="people"/>

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="person[@sex = 'f' and @smoker='yes']">
```

```
    <!-- Применить поведение по умолчанию, то есть копирование -->
    <xsl:apply-imports/>
  </xsl:template>

  <!-- Игнорировать прочих людей -->
  <xsl:template match="person"/>

</xsl:stylesheet>
```

Вместо этого можно написать единственный шаблон, который сопоставляется с теми элементами, которые нужно исключить, не применяя к ним никаких действий:

```
<xsl:template match="person[@sex != 'f' or @smoker != 'yes']" />
```

Обсуждение

Этот пример полезен тем, что позволяет сохранить структуру XML-документа, ничего не зная о том, какова эта структура. Нужно лишь знать, какие элементы следует отфильтровать, и понимать, что это можно сделать с помощью шаблонов.

Этот способ применим в таких контекстах, которые мало кто назвал бы запросами. Предположим, например, что нужно клонировать документ, заменив все атрибуты `sex` атрибутами `gender`:

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

  <xsl:import href="copy.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <xsl:template match="@sex">
    <xsl:attribute name="gender">
       <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

</xsl:stylesheet>
```

Элегантность этого решения в том, что оно работает для любого XML-документа вне зависимости от схемы. Если в документе есть элементы с атрибутом `sex`, он будет заменен на `gender`.

А сможете догадаться, что делает этот код?[1]

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

  <xsl:import href="copy.xslt"/>
```

[1] Он выводит оба атрибута gender и sex, но вы ведь и так уже догадались!

```
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="@sex">
      <xsl:attribute name="gender">
             <xsl:value-of select="."/>
      </xsl:attribute>
      <xsl:apply-imports/>
    </xsl:template>

</xsl:stylesheet>
```

9.5. Соединения

Задача

Требуется соотнести элементы в некотором документе с другими элементами в том же или ином документе.

Решение

Чтобы выполнить соединение, мы рассматриваем все возможные пары элементов (то есть берем декартово произведение) и оставляем только те, которые удовлетворяют критерию, по которому выполняется соединение (обычно это равенство).

Для демонстрации я адаптировал для XML базу данных о деталях, приведенную в книге Date *An Introduction to Database Systems* (Addison Wesley, 1986)[1]:

```
<database>
  <suppliers>
    <supplier id="S1" name="Smith" status="20" city="London"/>
    <supplier id="S2" name="Jones" status="10" city="Paris"/>
    <supplier id="S3" name="Blake" status="30" city="Paris"/>
    <supplier id="S4" name="Clark" status="20" city="London"/>
    <supplier id="S5" name="Adams" status="30" city="Athens"/>
  </suppliers>
  <parts>
    <part id="P1" name="Nut" color="Red" weight="12" city="Londo"/>
    <part id="P2" name="Bult" color="Green" weight="17" city="Paris"/>
    <part id="P3" name="Screw" color="Blue" weight="17" city="Rome"/>
    <part id="P4" name="Screw" color="Red" weight="14" city="London"/>
    <part id="P5" name="Cam" color="Blue" weight="12" city="Paris"/>
    <part id="P6" name="Cog" color="Red" weight="19" city="London"/>
  </parts>
  <inventory">
    <invrec sid="S1" pid="P1" qty="300"/>
```

[1] К. Дж. Дэйт «Введение в системы баз данных», издательство Вильямс, 2006.

```
    <invrec sid="S1" pid="P2" qty="200"/>
    <invrec sid="S1" pid="P3" qty="400"/>
    <invrec sid="S1" pid="P4" qty="200"/>
    <invrec sid="S1" pid="P5" qty="100"/>
    <invrec sid="S1" pid="P6" qty="100"/>
    <invrec sid="S2" pid="P1" qty="300"/>
    <invrec sid="S2" pid="P2" qty="400"/>
    <invrec sid="S3" pid="P2" qty="200"/>
    <invrec sid="S4" pid="P2" qty="200"/>
    <invrec sid="S4" pid="P4" qty="300"/>
    <invrec sid="S4" pid="P5" qty="400"/>
  </inventory>
</database>
```

Нам предстоит выполнить соединение, которое отвечает на вопрос: «Какие поставщики и детали находятся в одном и том же городе?»

К решению этой задачи на XSLT есть два основных подхода. Во-первых, можно использовать вложенные циклы `for-each`:

```
<xsl:template match="/">
  <result>
   <xsl:for-each select="database/suppliers/*">
    <xsl:variable name="supplier" select="."/>
    <xsl:for-each select="/database/parts/*[@city=current()/@city]">
       <colocated>
           <xsl:copy-of select="$supplier"/>
           <xsl:copy-of select="."/>
       </colocated>
    </xsl:for-each>
   </xsl:for-each>
  </result>
</xsl:template>
```

Второй способ – применить команду `apply-templates`:

```
<xsl:template match="/">
  <result>
   <xsl:apply-templates select="database/suppliers/supplier" />
  </result>
</xsl:template>

<xsl:template match="supplier">
  <xsl:apply-templates select="/database/parts/part[@city = current()/@city]">
    <xsl:with-param name="supplier" select="." />
  </xsl:apply-templates>
</xsl:template>
```

```
<xsl:template match="part">
    <xsl:param name="supplier" select="/.." />
    <colocated>
       <xsl:copy-of select="$supplier" />
       <xsl:copy-of select="." />
    </colocated>
</xsl:template>
```

Если в одном из соединяемых наборов очень много элементов, то для повышения производительности можно воспользоваться командой `xsl:key`:

```
<xsl:key name="part-city" match="part" use="@city"/>

<xsl:template match="/">
    <result>
       <xsl:for-each select="database/suppliers/*">
            <xsl:variable name="supplier" select="."/>
            <xsl:for-each select="key('part-city',$supplier/@city)">
            <colocated>
               <xsl:copy-of select="$supplier"/>
               <xsl:copy-of select="."/>
            </colocated>
            </xsl:for-each>
       </xsl:for-each>
    </result>
</xsl:template>
```

Обе таблицы стилей дают один и тот же результат:

```
<result>
   <colocated>
      <supplier id="S1" name="Smith" status="20" city="London"/>
      <part id="P1" name="Nut" color="Red" weight="12" city="London"/>
   </colocated>
   <colocated>
      <supplier id="S1" name="Smith" status="20" city="London"/>
      <part id="P4" name="Screw" color="Red" weight="14" city="London"/>
   </colocated>
   <colocated>
      <supplier id="S1" name="Smith" status="20" city="London"/>
      <part id="P6" name="Cog" color="Red" weight="19" city="London"/>
   </colocated>
   <colocated>
      <supplier id="S2" name="Jones" status="10" city="Paris"/>
      <part id="P2" name="Bult" color="Green" weight="17" city="Paris"/>
   </colocated>
```

```
    <colocated>
       <supplier id="S2" name="Jones" status="10" city="Paris"/>
       <part id="P5" name="Cam" color="Blue" weight="12" city="Paris"/>
    </colocated>
    <colocated>
       <supplier id="S3" name="Blake" status="30" city="Paris"/>
       <part id="P2" name="Bult" color="Green" weight="17" city="Paris"/>
    </colocated>
    <colocated>
       <supplier id="S3" name="Blake" status="30" city="Paris"/>
       <part id="P5" name="Cam" color="Blue" weight="12" city="Paris"/>
    </colocated>
    <colocated>
       <supplier id="S4" name="Clark" status="20" city="London"/>
       <part id="P1" name="Nut" color="Red" weight="12" city="London"/>
    </colocated>
    <colocated>
       <supplier id="S4" name="Clark" status="20" city="London"/>
       <part id="P4" name="Screw" color="Red" weight="14" city="London"/>
    </colocated>
    <colocated>
       <supplier id="S4" name="Clark" status="20" city="London"/>
       <part id="P6" name="Cog" color="Red" weight="19" city="London"/>
    </colocated>
</result>
```

Обсуждение

XSLT 1.0

Соединение, которое мы только что выполнили, называется *соединением по равенству* (equi-join), поскольку элементы сравнивались на равенство. В общем случае можно задавать и другие отношения. Рассмотрим, например, такой запрос: «Отобрать все сочетания поставщиков и деталей, в которых город, где находится поставщик, больше города, в котором производится деталь, в смысле лексикографического порядка».

Хорошо бы написать соответствующую таблицу стилей, как показано ниже, но в XSLT 1.0 не определены операции сравнения строк:

```
<xsl:template match="/">
  <result>
    <xsl:for-each select="database/suppliers/*">
       <xsl:variable name="supplier" select="."/>
         <!-- Это не работает! -->
       <xsl:for-each select="/database/parts/*[current( )/@city > @city]">
       <colocated>
```

```
            <xsl:copy-of select="$supplier"/>
            <xsl:copy-of select="."/>
      </colocated>
      </xsl:for-each>
    </xsl:for-each>
  </result>
</xsl:template>
```

Вместо этого придется создать таблицу с помощью команды `xsl:sort`, которая отобразит названия городов на целые числа, отражающие порядок следования. Здесь мы пользуемся умением процессора Saxon рассматривать переменные, которые содержат фрагменты результирующего дерева, как наборы узлов, если установлена версия 1.1. Но можно также воспользоваться функцией `node-set()`, если она поддерживается вашим процессором XSLT 1.0, или обратиться к процессору XSLT 2.0:

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:variable name="unique-cities"
    select="//@city[not(. = ../preceding::*/@city)]"/>

<xsl:variable name="city-ordering">
  <xsl:for-each select="$unique-cities">
    <xsl:sort select="."/>
    <city name="{.}" order="{position()}"/>
  </xsl:for-each>
</xsl:variable>

<xsl:template match="/">
  <result>
    <xsl:for-each select="database/suppliers/*">
    <xsl:variable name="s" select="."/>
    <xsl:for-each select="/database/parts/*">
       <xsl:variable name="p" select="."/>
       <xsl:if
            test="$city-ordering/*[@name = $s/@city]/@order &gt;
               $city-ordering/*[@name = $p/@city]/@order">
            <supplier-city-follows-part-city>
               <xsl:copy-of select="$s"/>
               <xsl:copy-of select="$p"/>
            </supplier-city-follows-part-city>
       </xsl:if>
    </xsl:for-each>
   </xsl:for-each>
 </result>
```

```
</xsl:template>

</xsl:stylesheet>
```

Этот запрос дает следующий результат:

```
<result>
   <supplier-city-follows-part-city>
      <supplier id="S2" name="Jones" status="10" city="Paris"/>
      <part id="P1" name="Nut" color="Red" weight="12" city="London"/>
   </supplier-city-follows-part-city>
   <supplier-city-follows-part-city>
      <supplier id="S2" name="Jones" status="10" city="Paris"/>
      <part id="P4" name="Screw" color="Red" weight="14" city="London"/>
   </supplier-city-follows-part-city>
   <supplier-city-follows-part-city>
      <supplier id="S2" name="Jones" status="10" city="Paris"/>
      <part id="P6" name="Cog" color="Red" weight="19" city="London"/>
   </supplier-city-follows-part-city>
   <supplier-city-follows-part-city>
      <supplier id="S3" name="Blake" status="30" city="Paris"/>
      <part id="P1" name="Nut" color="Red" weight="12" city="London"/>
   </supplier-city-follows-part-city>
   <supplier-city-follows-part-city>
      <supplier id="S3" name="Blake" status="30" city="Paris"/>
      <part id="P4" name="Screw" color="Red" weight="14" city="London"/>
   </supplier-city-follows-part-city>
   <supplier-city-follows-part-city>
      <supplier id="S3" name="Blake" status="30" city="Paris"/>
      <part id="P6" name="Cog" color="Red" weight="19" city="London"/>
   </supplier-city-follows-part-city>
</result>
```

XSLT 2.0

В XSLT 2.0 операторы сравнения корректно работают со строковыми значениями, поэтому годится и более простая таблица стилей:

```
<xsl:template match="/">
  <result>
    <xsl:for-each select="database/suppliers/*">
       <xsl:variable name="supplier" select="."/>
          <!-- В версии 2.0 это допустимо -->
       <xsl:for-each select="/database/parts/*[current( )/@city > @city]">
       <colocated>
           <xsl:copy-of select="$supplier"/>
           <xsl:copy-of select="."/>
```

```
        </colocated>
      </xsl:for-each>
    </xsl:for-each>
  </result>
</xsl:template>
```

9.6. Реализация на XSLT сценариев, приведенных в спецификации W3C XML Query

Задача

Требуется выполнить запрос, аналогичный одному из приведенных в качестве примеров в документе *http://www.w3.org/TR/2001/WD-xmlquery-use-cases-20011220*, но вместо языка XQuery (*http://www.w3.org/TR/xquery/*) использовать XSLT.

Решение

В следующих примерах приведены решения на XSLT для большинства сценариев, приведенных в документе консорциума W3C. Описания сценариев почти дословно взяты из этого документа.

1. Сценарий «XMP»: описание и образцы решения.
   В этом сценарии приведено несколько примеров запросов, иллюстрирующих требования, сформулированные сообществами разработчиков баз данных и средств обработки документов. Данные для запросов представлены в примерах 9.10 – 9.13.

Пример 9.10. bib.xml

```
<bib>
    <book year="1994">
       <title>TCP/IP Illustrated</title>
       <author><last>Stevens</last><first>W.</first></author>
       <publisher>Addison-Wesley</publisher>
       <price> 65.95</price>
    </book>

    <book year="1992">
       <title>Advanced Programming in the Unix environment</title>
       <author><last>Stevens</last><first>W.</first></author>
       <publisher>Addison-Wesley</publisher>
       <price>65.95</price>
    </book>

    <book year="2000">
       <title>Data on the Web</title>
```

```
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price> 39.95</price>
    </book>

    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
            <last>Gerbarg</last><first>Darcy</first>
            <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>

</bib>
```

Пример 9.11. reviews.xml

```
<reviews>
    <entry>
        <title>Data on the Web</title>
        <price>34.95</price>
        <review>
            A very good discussion of semi-structured database
            systems and XML.
        </review>
    </entry>
    <entry>
        <title>Advanced Programming in the Unix environment</title>
        <price>65.95</price>
        <review>
            A clear and detailed discussion of UNIX programming.
        </review>
    </entry>
    <entry>
        <title>TCP/IP Illustrated</title>
        <price>65.95</price>
        <review>
            One of the best books on TCP/IP.
        </review>
    </entry>
</reviews>
```

Пример 9.12. books.xml

```
<chapter>
    <title>Data Model</title>
    <section>
       <title>Syntax For Data Model</title>
   </section>
   <section>
       <title>XML</title>
       <section>
           <title>Basic Syntax</title>
       </section>
       <section>
           <title>XML and Semistructured Data</title>
       </section>
    </section>
</chapter>
```

Пример 9.13. prices.xml

```
<prices>
    <book>
       <title>Advanced Programming in the Unix environment</title>
       <source>www.amazon.com</source>
       <price>65.95</price>
    </book>
    <book>
       <title>Advanced Programming in the Unix environment </title>
       <source>www.bn.com</source>
       <price>65.95</price>
    </book>
    <book>
       <title> TCP/IP Illustrated </title>
       <source>www.amazon.com</source>
       <price>65.95</price>
    </book>
    <book>
       <title> TCP/IP Illustrated </title>
       <source>www.bn.com</source>
       <price>65.95</price>
    </book>
    <book>
       <title>Data on the Web</title>
       <source>www.amazon.com</source>
       <price>34.95</price>
```

```
    </book>
    <book>
       <title>Data on the Web</title>
       <source>www.bn.com</source>
       <price>39.95</price>
    </book>
</prices>
```

Вопрос 1. Перечислить книги в файле bib.xml, опубликованные издательством Addison-Wesley после 1991 года, указав для каждой год издания и название:

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

<xsl:import href="copy.xslt"/>

<xsl:template match="book[publisher = 'Addison-Wesley' and @year > 1991]">
    <xsl:copy-of select="."/>
</xsl:template>

<xsl:template match="book"/>

</xsl:stylesheet>
```

Вопрос 2. Создать плоский список всех пар название-автор из файла bib.xml, поместив каждую пару внутрь элемента result:

```
<xsl:template match="/">
<results>
    <xsl:apply-templates select="bib/book/author"/>
</results>
</xsl:template>

<xsl:template match="author">
    <result>
       <xsl:copy-of select="preceding-sibling::title"/>
       <xsl:copy-of select="."/>
    </result>
</xsl:template>
```

Вопрос 3. Для каждой книги в файле bib.xml вывести ее название и авторов, поместив их внутрь элемента result:

```
<xsl:template match="bib">
  <results>
    <xsl:for-each select="book">
       <result>
           <xsl:copy-of select="title"/>
           <xsl:copy-of select="author"/>
       </result>
    </xsl:for-each>
```

```
  </results>
</xsl:template>
```

Вопрос 4. Для всех авторов в файле bib.xml вывести имя автора и названия написанных им книг, поместив все это внутрь элемента result:

```
<xsl:template match="/">
<results>
 <xsl:for-each select="//author[not(.=preceding::author)]">
    <result>
    <xsl:copy-of select="."/>
    <xsl:for-each select="/bib/book[author=current()]">
       <xsl:copy-of select="title"/>
    </xsl:for-each>
   </result>
 </xsl:for-each>
</results>
```

Вопрос 5. Для каждой книги, найденной одновременно на сайтах http://www/bn.com (bib.xml) и http://www.amazon.com (reviews.xml), вывести название и цену, указанную в каждом источнике:

```
<xsl:variable name="bn" select="document('bib.xml')"/>
<xsl:variable name="amazon" select="document('reviews.xml')"/>

<!-- Решение 1 -->
<xsl:template match="/">
  <books-with-prices>
    <xsl:for-each select="$bn//book[title = $amazon//entry/title]">
    <book-with-prices>
       <xsl:copy-of select="title"/>
       <price-amazon><xsl:value-of
         select="$amazon//entry[title=current()/title]/price"/></price-amazon>
       <price-bn><xsl:value-of select="price"/></price-bn>
    </book-with-prices>
   </xsl:for-each>
 </books-with-prices>
</xsl:template>

<!-- Решение 2 -->
<xsl:template match="/">
  <books-with-prices>
  <xsl:for-each select="$bn//book">
    <xsl:variable name="bn-book" select="."/>
    <xsl:for-each select="$amazon//entry[title=$bn-book/title]">
       <book-with-prices>
           <xsl:copy-of select="title"/>
           <price-amazon><xsl:value-of select="price"/></price-amazon>
```

```
            <price-bn><xsl:value-of select="$bn-book/price"/></price-bn>
        </book-with-prices>
    </xsl:for-each>
   </xsl:for-each>
 </books-with-prices>
</xsl:template>
```

Вопрос 6. Для каждой книги, у которой есть хотя бы один автор, вывести название и имена первых двух авторов, а также пустой элемент "et-al", если у книги есть еще авторы:

```
<xsl:template match="bib">
  <xsl:copy>
    <xsl:for-each select="book[author]">
    <xsl:copy>
       <xsl:copy-of select="title"/>
       <xsl:copy-of select="author[position() &lt;= 2]"/>
       <xsl:if test="author[3]">
       <et-al/>
       </xsl:if>
    </xsl:copy>
   </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Вопрос 7. Перечислить названия и года издания всех книг, опубликованных издательством Addison-Wesley после 1991 года, в алфавитном порядке:

```
<xsl:template match="bib">
  <xsl:copy>
    <xsl:for-each select="book[publisher = 'Addison-Wesley'
           and @year > 1991]">
       <xsl:sort select="title"/>
       <xsl:copy>
          <xsl:copy-of select="@year"/>
          <xsl:copy-of select="title"/>
       </xsl:copy>
     </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Вопрос 8. В документе books.xml найти названия всех разделов (элементы section) и глав (элементы chapter), в которых встречается слово "XML", независимо от уровня вложенности:

```
<xsl:template match="/">
<results>
  <xsl:copy-of select="(//chapter/title |
  //section/title)[contains(.,'XML')]"/>
</results>
```

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

Вопрос 9. В документе prices.xml найти минимальную цену каждой книги и вывести ее в виде элемента "minprice", в котором название книги представлено атрибутом title:

```
<xsl:include href="../math/math.min.xslt"/>

<xsl:template match="/">
<results>
  <xsl:for-each select="//book/title[not(. = ./preceding::title)]">
   <xsl:variable name="min-price">
    <xsl:call-template name="math:min">
       <xsl:with-param name="nodes" select="//book[title =
                                        current()]/price"/>
    </xsl:call-template>
   </xsl:variable>
   <minprice title="{.}">
    <price><xsl:value-of select="$min-price"/></prices>
   </minprice>
  </xsl:for-each>
</results>
</xsl:template>
```

Вопрос 10. Для каждой книги, имеющей автора, вернуть ее название и всех авторов. Для каждой книги, имеющей редактора, вернуть элемент reference, содержащий название книги и название организации, к которой принадлежит редактор:

```
<xsl:template match="bib">
<xsl:copy>
  <xsl:for-each select="book[author]">
   <xsl:copy>
    <xsl:copy-of select="title"/>
    <xsl:copy-of select="author"/>
   </xsl:copy>
  </xsl:for-each>

  <xsl:for-each select="book[editor]">
   <reference>
    <xsl:copy-of select="title"/>
    <org><xsl:value-of select="editor/affiliation"/></org>
   </reference>
  </xsl:for-each>
</xsl:copy>
</xsl:template>
```

Вопрос 11. Найти пары книг, у которых разные названия, но один и тот же набор авторов (возможно, перечисленных в другом порядке):

```
<xsl:include href="query.equal-values.xslt"/>

<xsl:template match="bib">
  <xsl:copy>
   <xsl:for-each select="book[author]">
    <xsl:variable name="book1" select="."/>
    <xsl:for-each select="./following-sibling::book[author]">
       <xsl:variable name="same-authors">
            <xsl:call-template name="query:equal-values">
               <xsl:with-param name="nodes1" select="$book1/author"/>
               <xsl:with-param name="nodes2" select="author"/>
            </xsl:call-template>
       </xsl:variable>
       <xsl:if test="string($same-authors)">
            <book-pair>
               <xsl:copy-of select="$book1/title"/>
               <xsl:copy-of select="title"/>
            </book-pair>
       </xsl:if>
    </xsl:for-each>
   </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

2. Сценарий «TREE»: запросы, сохраняющие иерархию.
   Некоторые XML-документы обладают весьма гибкой структурой, в которой текст перемежается элементами, причем многие элементы необязательны. Структура документов может быть очень разнообразной. Обычно порядок следования и вложенности элементов имеет значение. Язык запросов XML должен иметь возможность извлекать элементы из документов с сохранением иерархии. Настоящий сценарий иллюстрирует это требования на примере гибкого документа Book.

В примерах 9.14 и 9.15 приведена DTD-схема и XML-данные, используемые в запросах.

Пример 9.14. book.dtd

```
<!ELEMENT book (title, author+, section+)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT section (title, (p | figure | section)* )>
    <!ATTLIST section
           id                    ID      #IMPLIED
           difficulty    CDATA   #IMPLIED>
    <!ELEMENT p (#PCDATA)>
```

```
    <!ELEMENT figure (title, image)>
    <!ATTLIST figure
           width          CDATA   #REQUIRED
           height         CDATA   #REQUIRED >
    <!ELEMENT image EMPTY>
    <!ATTLIST image
           source         CDATA   #REQUIRED >
```

Пример 9.15. books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Data on the Web</title>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
  <section id="intro" difficulty="easy" >
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
      <title>Audience</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Web Data and the Two Cultures</title>
      <p>Text ... </p>
      <figure height="400" width="400">
        <title>Traditional client/server architecture</title>
        <image source="csarch.gif"/>
      </figure>
      <p>Text ... </p>
    </section>
  </section>
  <section id="syntax" difficulty="medium" >
    <title>A Syntax For Data</title>
    <p>Text ... </p>
    <figure height="200" width="500">
      <title>Graph representations of structures</title>
      <image source="graphs.gif"/>
    </figure>
    <p>Text ... </p>
    <section>
      <title>Base Types</title>
      <p>Text ... </p>
```

```
    </section>
    <section>
      <title>Representing Relational Databases</title>
      <p>Text ... </p>
      <figure height="250" width="400">
        <title>Examples of Relations</title>
        <image source="relations.gif"/>
      </figure>
    </section>
    <section>
      <title>Representing Object Databases</title>
      <p>Text ... </p>
    </section>
  </section>
</book>
```

Вопрос 1. Подготовить (иерархическое) оглавление книги Book1, перечислив все разделы вместе с названиями. Сохранить все атрибуты каждого элемента <section>, если таковые присутствуют:

```
<xsl:template match="book">
  <toc>
    <xsl:apply-templates/>
  </toc>
</xsl:template>

<!-- Копировать элемент toc -->
<xsl:template match="section | section/title | section/title/text()">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>

<!-- Подавить все прочие элементы -->
<xsl:template match="* | text()"/>
```

Вопрос 2. Подготовить (плоский) список рисунков для книги Book1, перечислив все рисунки вместе с названием. Сохранить все атрибуты каждого элемента <figure>, если таковые присутствуют:

```
<xsl:template match="book">
  <figlist>
    <xsl:for-each select=".//figure">
          <xsl:copy>
                <xsl:copy-of select="@*"/>
                <xsl:copy-of select="title"/>
          </xsl:copy>
    </xsl:for-each>
```

```
  </figlist>
</xsl:template>
```

Вопрос 3. Сколько в книге Book1 разделов и сколько рисунков?

```
<xsl:template match="/">
  <section-count><xsl:value-of select="count(//section)"/></section-count>
  <figure-count><xsl:value-of select="count(//figure)"/></figure-count>
</xsl:template>
```

Вопрос 4. Сколько в книге Book1 разделов верхнего уровня?

```
<xsl:template match="book">
    <top_section_count>
        <xsl:value-of select="count(section)"/>
    </top_section_count>
</xsl:template>
```

Вопрос 5. Создать плоский список элементов section в книге Book1. Вместо исходных атрибутов у каждого элемента section должно быть два атрибута: название раздела и количество рисунков, содержащихся в данном разделе без учета подразделов:

```
<xsl:template match="book">
<section_list>
  <xsl:for-each select=".//section">
    <section title="{title}" figcount="{count(figure)}"/>
  </xsl:for-each>
</section_list>
</xsl:template>
```

Вопрос 6. Создать иерархический список элементов section в книге Book1, сохранив исходные атрибуты и иерархию. Внутрь каждого элемента section поместить название раздела и элемент, содержащий количество рисунков в нем без учета подразделов. См. примеры 9.16 и 9.17:

Пример 9.16. Решение, вытекающее из моей интерпретации условия

```
<xsl:template match="book">
<toc>
    <xsl:apply-templates select="section"/>
</toc>
</xsl:template>

<xsl:template match="section">
    <xsl:copy>
           <xsl:copy-of select="@*"/>
           <xsl:copy-of select="title"/>
           <figcount><xsl:value-of select="count(figure)"/></figcount>
           <xsl:apply-templates select="section"/>
    </xsl:copy>
</xsl:template>
```

Пример 9.17. Что имел в виду W3C, исходя из приведенного результата и запроса на языке XQuery

```
<xsl:template match="book">
<toc>
    <xsl:for-each select="//section">
        <xsl:sort select="count(ancestor::section)"/>
        <xsl:apply-templates select="."/>
    </xsl:for-each>
</toc>
</xsl:template>

<xsl:template match="section">
    <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:copy-of select="title"/>
        <figcount><xsl:value-of select="count(figure)"/></figcount>
        <xsl:apply-templates select="section"/>
    </xsl:copy>
</xsl:template>
```

3. Сценарий «SEQ»: запросы, основанные на последовательности.
   В этом сценарии иллюстрируются запросы, основанные на последовательности, в которой элементы появляются в документе. Хотя в большинстве традиционных СУБД и объектных систем последовательность не играет роли, в структурированных документах она может иметь значение. Ниже представлены запросы к медицинскому отчету.

```
<!DOCTYPE report [
  <!ELEMENT report (section*)>
  <!ELEMENT section (section.title, section.content)>
  <!ELEMENT section.title  (#PCDATA )>
  <!ELEMENT section.content  (#PCDATA | anesthesia | prep
                            | incision | action | observation )*>
  <!ELEMENT anesthesia (#PCDATA)>
  <!ELEMENT prep ( (#PCDATA | action)* )>
  <!ELEMENT incision ( (#PCDATA | geography | instrument)* )>
  <!ELEMENT action ( (#PCDATA | instrument )* )>
  <!ELEMENT observation (#PCDATA)>
  <!ELEMENT geography (#PCDATA)>
  <!ELEMENT instrument (#PCDATA)>
]>
<report>
  <section>
    <section.title>Процедура</section.title>
     <section.content>
```

```
        Пациентку доставили в операционную, уложили на спину и провели
        <anesthesia>вводную общую анестезию.</anesthesia>
        <prep>
          <action>Для опорожнения мочевого пузыря в него
           ввели катетер Фоли, после чего кожу живота
           обработали антисептическим раствором и драпировали
           стерильной тканью.
        </prep>
        <incision>Произвели дугообразный разрез
          <geography> по средней линии живота ниже пупка</geography>
          и рассекли подкожную клетчатку
          <instrument>электрокаутером.</instrument>
        </incision>
        По обнаружении фасции
        <action>с обеих сторон от средней линии был наложен шов-держалка
        Максон #2 0.
        </action>
        <incision>
          После рассечения фасции
          <instrument>электрокаутером</instrument>
          вскрыли брюшину.
        </incision>
         <observation>По обнаружении тонкой кишки</observation>
        <action>
            под визуальным контролем ввели
          <instrument>троакар Хассона</instrument>.
        </action>
        <action>
          <instrument>Троакар</instrument>
            имплантировали в фасцию при помощи шва-держалки.
        </action>
       </section.content>
    </section>
</report>
```

Вопрос 1. Какие инструменты были использованы для второго разреза (incision) в разделе Procedure документа Report1?

```
<xsl:template match="section[section.title = 'Procedure']">
    <xsl:copy-of select="(.//incision)[2]/instrument"/>
</xsl:template>
```

Вопрос 2. Исходя из раздела "Процедура" документа Report1, какие первые два инструмента следует использовать?

```
<xsl:template match="section[section.title = 'Процедура']">
    <xsl:copy-of select="(.//instrument)[position() &lt;= 2]"/>
</xsl:template>
```

Вопрос 3. Какие инструменты были использованы в первых двух действиях после второго разреза?

```
<xsl:template match="report">
<!-- i2 = второй элемент incision в отчете -->
<xsl:variable name="i2" select="(.//incision)[2]"/>
<!-- Для всех действий (action), следующих за i2,
     выбрать инструменты, использованные в первых двух -->
<xsl:copy-of
    select="($i2/following::action)[position() &lt;= 2]/instrument"/>
</xsl:template>
```

Вопрос 4. Найти все разделы "Процедура", в которых первому элементу incision не предшествует ни один элемент anesthesia:

```
<xsl:template match="section[section.title = 'Процедура']">
    <xsl:variable name="i1" select="(.//incision)[1]"/>
    <xsl:if test=".//anesthesia[preceding::incision = $i1]">
           <xsl:copy-of select="current()"/>
    </xsl:if>
</xsl:template>
```

Если результат этого запроса не пуст, то не за горами судебное разбирательство!

Вопрос 5. Что происходило между первым и вторым разрезами?

```
<xsl:template match="report">
<critical_sequence>
    <!-- i1 = первый элемент incision в отчете -->
    <xsl:variable name="i1" select="(.//incision)[1]"/>
    <!-- i2 = второй элемент incision в отчете -->
    <xsl:variable name="i2" select="(.//incision)[2]"/>
    <!-- скопировать все узлы-братья, следующие за i1, которым
    не предшествует элемент i2 и которые сами не совпадают с i2 -->
    <xsl:for-each select="$i1/following-sibling::node()
       [not(./preceding::incision = $i2) and not(. = $i2)]">
       <xsl:copy-of select="."/>
    </xsl:for-each>
</critical_sequence>
</xsl:template>
```

В вопросах 4 и 5 я предполагаю, что строковые значения всех элементов incision различны. Для данных из приведенного примера это справедливо, но в общем случае может быть и не так. Чтобы получить правильное решение, воспользуйтесь рецептом 4.2. Так, в вопросе 4 проверка должна выглядеть так:

```
test=".//anesthesia[count(./preceding::incision |
$i1) = count(./preceding::incision)]"
```

4. Сценарий «R»: доступ к реляционным данным.
   Одно из важных применений языка запросов к XML – это доступ к данным, хранящимся в реляционной базе. В настоящем сценарии описывается один из возможных способов реализации такого доступа. В реляционной СУБД может быть определено представление, в котором каждая таблица (отношение) принимает форму XML-документа. Например, документный элемент может представлять таблицу в целом, а каждая ее строка (кортеж) представляется вложенным элементом. В элементы, соответствующие строкам, в свою очередь вложены элементы, представляющие колонки. Колонкам, допускающим значения null, соответствуют необязательные элементы; если такой элемент опущен, значит, соответствующая ему колонка содержит null.
   Рассмотрим, например, реляционную базу для онлайнового аукциона. В ней есть таблица USERS, в которой хранится информация о зарегистрированных пользователях, для каждого из которых определен уникальный идентификатор. Пользователь может либо выставлять товар на аукцион, либо принимать участие в торгах. В таблице ITEMS хранятся торгуемые сейчас или недавно выставленные на торги товары, причем в каждой ее строке записан идентификатор пользователя, предложившего товар. В таблице BIDS хранится информация о торгах: идентификатор торгующегося пользователя и идентификатор торгуемого товара.
   Поскольку количество запросов в этом сценарии велико, мы реализуем только часть. Реализация остальных запросов будет неплохим упражнением для тех, кто хотел бы отточить свое владение XSLT. См. примеры 9.18 – 9.20.

Пример 9.18. users.xml

```
<users>
  <user_tuple>
    <userid>U01</userid>
    <name>Tom Jones</name>
    <rating>B</rating>
  </user_tuple>
  <user_tuple>
    <userid>U02</userid>
    <name>Mary Doe</name>
    <rating>A</rating>
  </user_tuple>
  <user_tuple>
    <userid>U03</userid>
    <name>Dee Linquent</name>
    <rating>D</rating>
  </user_tuple>
  <user_tuple>
    <userid>U04</userid>
```

```
    <name>Roger Smith</name>
    <rating>C</rating>
  </user_tuple>
  <user_tuple>
    <userid>U05</userid>
    <name>Jack Sprat</name>
    <rating>B</rating>
  </user_tuple>
  <user_tuple>
    <userid>U06</userid>
    <name>Rip Van Winkle</name>
    <rating>B</rating>
  </user_tuple>
</users>
```

Пример 9.19. items.xml

```
<items>
  <item_tuple>
    <itemno>1001</itemno>
    <description>Red Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-01-05</start_date>
    <end_date>99-01-20</end_date>
    <reserve_price>40</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1002</itemno>
    <description>Motorcycle</description>
    <offered_by>U02</offered_by>
    <start_date>99-02-11</start_date>
    <end_date>99-03-15</end_date>
    <reserve_price>500</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1003</itemno>
    <description>Old Bicycle</description>
    <offered_by>U02</offered_by>
    <start_date>99-01-10</start_date>
    <end_date>99-02-20</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1004</itemno>
```

```
    <description>Tricycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-02-25</start_date>
    <end_date>99-03-08</end_date>
    <reserve_price>15</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1005</itemno>
    <description>Tennis Racket</description>
    <offered_by>U03</offered_by>
    <start_date>99-03-19</start_date>
    <end_date>99-04-30</end_date>
    <reserve_price>20</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1006</itemno>
    <description>Helicopter</description>
    <offered_by>U03</offered_by>
    <start_date>99-05-05</start_date>
    <end_date>99-05-25</end_date>
    <reserve_price>50000</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1007</itemno>
    <description>Racing Bicycle</description>
    <offered_by>U04</offered_by>
    <start_date>99-01-20</start_date>
    <end_date>99-02-20</end_date>
    <reserve_price>200</reserve_price>
  </item_tuple>
  <item_tuple>
    <itemno>1008</itemno>
    <description>Broken Bicycle</description>
    <offered_by>U01</offered_by>
    <start_date>99-02-05</start_date>
    <end_date>99-03-06</end_date>
    <reserve_price>25</reserve_price>
  </item_tuple>
</items>
```

Пример 9.20. bids.xml

```
<bids>
  <bid_tuple>
```

```
  <userid>U02</userid>
  <itemno>1001</itemno>
  <bid> 35</bid>
  <bid_date>99-01-07 </bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U04</userid>
  <itemno>1001</itemno>
  <bid>40</bid>
  <bid_date>99-01-08</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U02</userid>
  <itemno>1001 </itemno>
  <bid>45</bid>
  <bid_date>99-01-11</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U04</userid>
  <itemno>1001</itemno>
  <bid>50</bid>
  <bid_date>99-01-13</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U02</userid>
  <itemno>1001</itemno>
  <bid>55</bid>
  <bid_date>99-01-15</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U01</userid>
  <itemno>1002</itemno>
  <bid>400</bid>
  <bid_date>99-02-14</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U02</userid>
  <itemno>1002</itemno>
  <bid>600</bid>
  <bid_date>99-02-16</bid_date>
</bid_tuple>
<bid_tuple>
  <userid>U03</userid>
  <itemno>1002</itemno>
```

```
    <bid>800</bid>
    <bid_date>99-02-17</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1002</itemno>
    <bid>1000</bid>
    <bid_date>99-02-25</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1002</itemno>
    <bid>1200</bid>
    <bid_date>99-03-02</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1003</itemno>
    <bid>15</bid>
    <bid_date>99-01-22</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U05</userid>
    <itemno>1003</itemno>
    <bid>20</bid>
    <bid_date>99-02-03</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U01</userid>
    <itemno>1004</itemno>
    <bid>40</bid>
    <bid_date>99-03-05</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U03</userid>
    <itemno>1007</itemno>
    <bid>175</bid>
    <bid_date>99-01-25</bid_date>
  </bid_tuple>
  <bid_tuple>
    <userid>U05</userid>
    <itemno>1007</itemno>
    <bid>200</bid>
    <bid_date>99-02-08</bid_date>
```

```
  </bid_tuple>
  <bid_tuple>
    <userid>U04</userid>
    <itemno>1007</itemno>
    <bid>225</bid>
    <bid_date>99-02-12</bid_date>
  </bid_tuple>
</bids>
```

Вопрос 1. Перечислить идентификаторы и описания всех велосипедов (bicycle), по которым сейчас идут торги, упорядочив список по идентификатору:

```
<xsl:include href="../date/date.date-time.xslt"/>

<!-- Чтобы результат был похож на приведенный в документе W3C -->
<xsl:param name="today" select="'1999-01-21'"/>

<xsl:template match="items">

    <xsl:variable name="today-abs">
       <xsl:call-template name="date:date-to-absolute-day">
       <xsl:with-param name="date" select="$today"/>
       </xsl:call-template>
    </xsl:variable>

<result>
    <xsl:for-each select="item_tuple">
       <xsl:sort select="itemno" data-type="number"/>

       <xsl:variable name="start-abs">
       <xsl:call-template name="date:date-to-absolute-day">
         <xsl:with-param name="date" select="start_date"/>
       </xsl:call-template>
       </xsl:variable>

       <xsl:variable name="end-abs">
       <xsl:call-template name="date:date-to-absolute-day">
         <xsl:with-param name="date" select="end_date"/>
       </xsl:call-template>
       </xsl:variable>

       <xsl:if test="$start-abs &lt;= $today-abs and $end-abs >=
             $today-abs and contains(description, 'Bicycle')">
         <xsl:copy>
          <xsl:copy-of select="itemno"/>
          <xsl:copy-of select="description"/>
```

```
            </xsl:copy>
        </xsl:if>

    </xsl:for-each>
</result>
</xsl:template>
```

Вопрос 2. Для всех велосипедов вывести идентификатор товара, его описание и наивысшее предложение (если есть хотя бы одно), упорядочив список по идентификатору:

```
<xsl:include href="../math/math.max.xslt"/>

<xsl:template match="items">

<result>
  <xsl:for-each select="item_tuple[contains(description,'Bicycle')]">
   <xsl:sort select="itemno" data-type="number"/>

   <xsl:variable name="bids"
    select="document('bids.xml')//bid_tuple[itemno=current()/itemno]/bid"/>

    <xsl:variable name="high-bid">
    <xsl:call-template name="math:max">
           <xsl:with-param name="nodes" select="$bids"/>
    </xsl:call-template>
   </xsl:variable>

   <xsl:copy>
    <xsl:copy-of select="itemno"/>
    <xsl:copy-of select="description"/>
    <high_bid><xsl:if test="$bids"><xsl:value-of
           select="$high-bid"/></xsl:if></high_bid>
  </xsl:copy>

  </xsl:for-each>
</result>
</xsl:template>
```

Вопрос 3. Найти все случаи, когда пользователь с рейтингом ниже (то есть больше в алфавитном порядке) "C" предлагает товар с зарезервированной ценой более 1000:

```
<!-- Строго говоря, необязательно, но в условии не определена система
рейтингования, поэтому выводим ее динамически! -->
<xsl:variable name="ratings">
  <xsl:for-each select="document('users.xml')//user_tuple/rating">
   <xsl:sort select="." data-type="text"/>
```

```
   <xsl:if test="not(. = ./preceding::rating)">
    <xsl:value-of select="."/>
   </xsl:if>
  </xsl:for-each>
</xsl:variable>

<xsl:template match="items">
<result>
  <xsl:for-each select="item_tuple[reserve_price > 1000]">

   <xsl:variable name="user" select="document('users.xml')//user_tuple[userid
    = current()/offered_by]"/>

   <xsl:if test="string-length(substring-before($ratings,$user/rating)) >
   string-length(substring-before($ratings,'C'))">
    <warning>
           <xsl:copy-of select="$user/name"/>
           <xsl:copy-of select="$user/rating"/>
           <xsl:copy-of select="description"/>
           <xsl:copy-of select="reserve_price"/>
    </warning>
   </xsl:if>
 </xsl:for-each>
</result>
</xsl:template>
```

Вопрос 4. Вывести идентификаторы и описания всех товаров, по которым нет торгов:

```
<xsl:template match="items">
<result>
  <xsl:for-each select="item_tuple">

   <xsl:if test="not(document('bids.xml')//bid_tuple[itemno =
   current()/itemno])">
    <no_bid_item>
           <xsl:copy-of select="itemno"/>
           <xsl:copy-of select="description"/>
     </no_bid_item>


  </xsl:for-each>
</result>
</xsl:template>
```

5. Сценарий «SGML»: язык Standard Generalized Markup Language.
   Документ и запросы для этого сценария были подготовлены для конференции 1992 года по языку Standard Generalized Markup Language (SGML).

Удобства ради определение типа документа (DTD-схема) и сам пример документа переведены с SGML на XML.

В настоящей главе эти запросы не приводятся, поскольку они мало чем отличаются от рассмотренных в других сценариях.

6. Сценарий «TEXT»: полнотекстовый поиск.

   В этом сценарии мы будем рассматривать профили компаний и новостные документы, содержащие информацию для отделов по связям с общественностью и касательно слияний и поглощений. Иллюстрируется несколько запросов для поиска текста в документах и различные способы представления результатов, полученных в результате сопоставления с профилем компании и содержанием новостей.

   При поиске компании по названию учитываются отдельные слова. Эти слова могут встречаться в названии в любом регистре и разделяться любыми символами пропуска.

   Все запросы можно выразить на XSLT 1.0. Однако при этом потребуются достаточно сложные механизмы текстового поиска. Например, для большинства сложных запросов нужно уметь проверять, входит ли в строку любое слово из заданного набора. Кроме того, для многих запросов необходимо учитывать разбиение текста на смысловые единицы, например, предложения.

   Принимая во внимание приемы, описанные в главе 1, должно быть понятно, что решать такие задачи на XSLT возможно. Однако, придется запастись достаточно общей библиотекой утилит для поиска по тексту. Разработке подобных библиотек посвящена глава 14, в которой мы еще вернемся к некоторым наиболее сложным запросам для полнотекстового поиска. А пока решим две самых простых задачи из сформулированных в документе W3C.

Вопрос 1. Найти все новости, в заголовке которые встречается название "Foobar Corporation":

```
<xsl:template match="news">
<result>
  <xsl:copy-of select="news_item/title[contains(., 'Foobar Corporation')]"/>
</result>
</xsl:template>
```

Вопрос 2. Для каждой новости, относящейся к компании Gorilla Corporation, создать элемент "item_summary", содержащие краткое резюме: название, дату и первый абзац новости, разделенные точками. Новость считается релевантной, если название компании упоминается в любом месте содержимого новости:

```
<xsl:template match="news">
<result>
  <xsl:for-each select="news_item[contains(content,'Gorilla Corporation')]">
    <item_summary>
     <xsl:value-of select="normalize-space(title)"/>. <xsl:text/>
```

```
    <xsl:value-of select="normalize-space(date)"/>. <xsl:text/>
    <xsl:value-of select="normalize-space(content/par[1])"/>
   </item_summary>
  </xsl:for-each>
</result>
</xsl:template>
```

7. Сценарий «PARTS»: рекурсивная деталировка.

   В этом сценарии показано, как с помощью рекурсивного запроса можно построить иерархический документ произвольной глубины из плоских структур, хранящихся в базе данных.

   В качестве примера взята база данных, содержащая информацию о том, как одни детали собираются из других.

   На вход подается «плоский» документ, в котором каждая деталь представлена элементом `<part>`, имеющим атрибуты `partid` и `name`. Каждая деталь может входить или в более крупную деталь, в таком случае идентификатор (partid) более крупной детали задается в атрибуте `partof`. Рассматриваемый документ можно было бы получить из базы данных, в которой каждая деталь представлена строкой таблицы с первичным ключом `partid` и внешним ключом `partof`, ссылающимся на `partid`.

   Сложность заключается в том, чтобы написать запрос, который преобразует «плоское» представление деталировки, основанное на внешних ключах, в иерархическое, где информация о составе сборок представлена самой структурой документа.

   Входные данные описываются следующей DTD-схемой:

```
<!DOCTYPE partlist [
    <!ELEMENT partlist (part*)>
    <!ELEMENT part EMPTY>
    <!ATTLIST part
            partid CDATA   #REQUIRED
            partof CDATA   #IMPLIED
            name   CDATA   #REQUIRED>
]>
```

   Хотя атрибуты `partid` и `partof` могли бы иметь тип ID и IDREF соответственно, в приведенной выше схеме мы трактуем их как символьные данные, которые, возможно, были материализованы непосредственно из реляционной базы. Каждый атрибут `partof` соответствует в точности одному `partid`. Детали, не имеющие атрибута `partof`, не входят ни в какую сборку.

   Выходные данные описываются следующей DTD-схемой:

```
<!DOCTYPE parttree [
    <!ELEMENT parttree (part*)>
    <!ELEMENT part (part*)>
    <!ATTLIST part
```

```
            partid CDATA   #REQUIRED
            name   CDATA   #REQUIRED>
]>
```

Данные, описываемые такой схемой, могли бы выглядеть следующим образом:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<partlist>
    <part partid="0" name="car"/>
    <part partid="1" partof="0" name="engine"/>
    <part partid="2" partof="0" name="door"/>
    <part partid="3" partof="1" name="piston"/>
    <part partid="4" partof="2" name="window"/>
    <part partid="5" partof="2" name="lock"/>
    <part partid="10" name="skateboard"/>
    <part partid="11" partof="10" name="board"/>
    <part partid="12" partof="10" name="wheel"/>
    <part partid="20" name="canoe"/>
</partlist>
```

Вопрос 1. Преобразовать документ из формата "partlist" в формат "parttree" (см. описание DTD выше). В результирующем документе вхождение одной детали в другую представлено вложенностью соответствующих элементов <part>. Детали, не являющиеся частью более крупных деталей, должны быть представлены отдельными элементами верхнего уровня:

```
<xsl:template match="partlist">
    <parttree>
       <!-- Начать с деталей, не входящих в более крупные сборки -->
    <xsl:apply-templates select="part[not(@partof)]"/>
  </parttree>
</xsl:template>

<xsl:template match="part">
  <part partid="{@partid}" name="{@name}">
    <xsl:apply-templates select="../part[@partof = current( )/@partid]"/>
  </part>
</xsl:template>
```

Как выясняется, такие преобразования на XSLT выглядят даже проще, чем на XQuery. Вот, например, решение той же задачи на языке XQuery, приведенное в документе W3C:

```
define function one_level (element $p) returns element
{
    <part partid="{ $p/@partid }"
          name="{ $p/@name }" >
        {
            for $s in document("partlist.xml")//part
```

```
            where $s/@partof = $p/@partid
            return one_level($s)
        }
    </part>
}
<parttree>
  {
    for $p in document("partlist.xml")//part[empty(@partof)]
    return one_level($p)
  }
</parttree>
```

Даже не зная языка XQuery, легко видеть, что на нем рекурсию приходится реализовывать явно, тогда как в XSLT конструкция apply-templates позволяет записать решение более декларативно. Согласен, разница не так уж велика, но все же мне кажется, что для таких задач решение на XSLT элегантнее.

1. Сценарий «REF»: запросы, основанные на ссылках[1].
   Ссылки – важный аспект XML. В этом сценарии описывается база данных, в которой ссылки играют заметную роль, и приводится несколько репрезентативных запросов, где эти ссылки используются.
   Предположим, что в файле *census.xml* имеются элементы для всех людей, участвовавших в последней переписи. Для каждого элемента person в атрибутах записаны имя (name), профессия (job) и ссылка на супруга/супругу (spouse), если таковой имеется. `spouse` – это атрибут типа IDREF, соответствующий атрибуту `name` типа ID в элементе, описывающем супруга (супругу).
   Отношение родитель-ребенок кодируется вложенностью элементов `person`. Иначе говоря, элемент, представляющий ребенка, содержится внутри элемента, представляющего его отца или мать. Из-за возможности смерти, развода или повторного брака ребенок может быть ассоциирован либо с отцом, либо с матерью (но не с тем и другим одновременно). В данном упражнении фраза «дети X» подразумевает также «детей супруга(и) X». Например, если Джо и Марта – супруги, при этом элемент Джо содержит элемент Сэм, а элемент Марта содержит элемент Дэйв, то детьми Джо и Марты считаются как Сэм, так и Дэйв. У каждого человека, участвовавшего в переписи, может быть 0, 1 или 2 родителя.

В этом сценарии мы будем использовать входной документ *census.xml*, описываемый следующей DTD-схемой:

```
<!DOCTYPE census [
    <!ELEMENT census (person*)>
    <!ELEMENT person (person*)>
    <!ATTLIST person
```

[1] Эти запросы исключены из последней редакции документа W3C.

```
            name    ID      #REQUIRED
            spouse  IDREF   #IMPLIED
            job     CDATA   #IMPLIED >
]>
```

В следующем фрагменте описываются две семьи, члены которых связаны несколькими перекрестными браками:

```
<census>
  <person name="Bill" job="Teacher">
   <person name="Joe" job="Painter" spouse="Martha">
    <person name="Sam" job="Nurse">
       <person name="Fred" job="Senator" spouse="Jane">
       </person>
    </person>
    <person name="Karen" job="Doctor" spouse="Steve">
    </person>
   </person>
   <person name="Mary" job="Pilot">
   <person name="Susan" job="Pilot" spouse="Dave">
    </person>
   </person>
  </person>
  <person name="Frank" job="Writer">
   <person name="Martha" job="Programmer" spouse="Joe">
    <person name="Dave" job="Athlete" spouse="Susan">
    </person>
   </person>
   <person name="John" job="Artist">
    <person name="Helen" job="Athlete">
    </person>
    <person name="Steve" job="Accountant" spouse="Karen">
       <person name="Jane" job="Doctor" spouse="Fred">
       </person>
    </person>
   </person>
  </person>
</census>
```

Вопрос 1. Найти супруга Марты:

```
<xsl:strip-space elements="*"/>

<xsl:template match="person[@spouse='Martha']">
  <xsl:copy>
   <xsl:copy-of select="@*"/>
  </xsl:copy>
```

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

Вопрос 2. Найти детей атлетов:

```
<xsl:template match="census">
  <xsl:variable name="everyone"  select="//person"/>
  <result>
      <!-- Для каждого человека, имеющего детей -->
   <xsl:for-each select="$everyone[person]">
    <xsl:variable name="spouse"
         select="$everyone[@spouse=current( )/@name]"/>
    <xsl:if test="./person/@job = 'Athlete' or
                  $spouse/person/@job = 'Athlete'">
        <xsl:copy>
           <xsl:copy-of select="@*"/>
        </xsl:copy>
    </xsl:if>
   </xsl:for-each>
  </result>
</xsl:template>
```

Вопрос 3. Найти всех людей, имеющих ту же профессию, что у одного из родителей:

Оставляю для самостоятельного упражнения.

Вопрос 4. Вывести имена родителей и детей, имеющих одну и ту же профессию, и включить в отчет саму профессию:

```
<xsl:template match="census">
  <xsl:variable name="everyone"  select="//person"/>
  <result>
      <!-- Для каждого человека, имеющего детей -->
    <xsl:for-each select="$everyone[person]">

        <xsl:variable name="spouse"
           select="$everyone[@spouse=current( )/@name]"/>

       <xsl:apply-templates select="person[@job = current( )/@job]">
          <xsl:with-param name="parent" select="@name"/>
       </xsl:apply-templates>

       <xsl:apply-templates select="person[@job = $spouse/@job]">
          <xsl:with-param name="parent" select="$spouse/@name"/>
       </xsl:apply-templates>

   </xsl:for-each>
  </result>
</xsl:template>
```

```
<xsl:template match="person">
  <xsl:param name="parent"/>
  <match parent="{$parent}"  child="{@name}" job="{@job}"/>
</xsl:template>
```

Вопрос 5. Вывести пары имен прадедушка (прабабушка)-правнук (правнучка):

```
<xsl:template match="census">
  <xsl:variable name="everyone"  select="//person"/>
  <result>
   <!-- Для каждого правнука -->
   <xsl:for-each select="$everyone[../../../person]">
    <!-- Получить прародителя1 -->
    <grandparent name="{../../@name}" grandchild="{@name}"/>
    <!-- Получить прародителя2, если супруг(а) прародителя1 переписан -->
    <xsl:if test="../../@spouse">
       <grandparent name="{../../@spouse}" grandchild="{@name}"/>
    </xsl:if>
    <!-- Получить имена супруга(и) родителя этого человека
    (то есть отца или матери) -->
    <xsl:variable name="spouse-of-parent" select="../@spouse"/>
    <!-- Получить родителей супруга родителя, если переписан -->
    <xsl:variable name="gp3"
       select="$everyone[person/@name=$spouse-of-parent]"/>
    <xsl:if test="$gp3">
       <grandparent name="{$gp3/@name}" grandchild="{@name}"/>
       <xsl:if test="$gp3/@spouse">
           <grandparent name="{$gp3/@spouse}" grandchild="{@name}"/>
       </xsl:if>
    </xsl:if>
   </xsl:for-each>
  </result>
</xsl:template>
```

Вопрос 6. Найти всех бездетных:

```
<xsl:strip-space elements="*"/>
<xsl:template match="census">
  <xsl:variable name="everyone"  select="//person"/>
  <result>
   <xsl:for-each select="$everyone[not(./person)]">
    <xsl:variable name="spouse"
       select="$everyone[@name = current( )/@spouse]"/>
    <xsl:if test="not ($spouse) or not($spouse/person)">
       <xsl:copy-of select="."/>
    </xsl:if>
    </xsl:for-each>
  </result>
```

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

Вопрос 7. Вывести имена всех потомков Джо. Каждого потомка представить элементом, в котором имя является содержимым, а семейное положение и количество детей — атрибутами. Отсортировать потомков сначала по убыванию количества детей, а потом по имени в алфавитном порядке:
<xsl:variable name="everyone" select="//person"/>

```
<xsl:template match="census">
  <result>
    <xsl:apply-templates select="//person[@name='Joe']"/>
  </result>
</xsl:template>

<xsl:template match="person">

  <xsl:variable name="all-desc">
    <xsl:call-template name="descendants">
    <xsl:with-param name="nodes" select="."/>
    </xsl:call-template>
  </xsl:variable>

  <xsl:for-each select="exsl:node-set($all-desc)/*">
    <xsl:sort select="count(./* | $everyone[@name = current( )/@spouse]/*)"
    order="descending" data-type="number"/>
     <xsl:sort select="@name"/>
     <xsl:variable name="mstatus"
        select="normalize-space(
            substring('No Yes',boolean(@spouse)* 3+1,3))"/>
     <person married="{$mstatus}"
        nkids="{count(./* | $everyone[@name = current( )/@spouse]/*)}">
        <xsl:value-of select="@name"/>
   </person>
  </xsl:for-each>
</xsl:template>

<xsl:template name="descendants">
  <xsl:param name="nodes"/>
  <xsl:param name="descendants" select="/.."/>

  <xsl:choose>
   <xsl:when test="not($nodes)">
    <xsl:copy-of select="$descendants"/>
   </xsl:when>
   <xsl:otherwise>
```

```
    <xsl:call-template name="descendants">
        <xsl:with-param name="nodes" select="$nodes[position( ) > 1] |
          $nodes[1]/person | id($nodes[1]/@spouse)/person"/>
        <xsl:with-param name="descendants" select="$descendants |
          $nodes[1]/person | id($nodes[1]/@spouse)/person"/>
    </xsl:call-template>
   </xsl:otherwise>
  </xsl:choose>

</xsl:template>
```

Задача решена, но как некрасиво! Сложность связана с тем, что для сортировки приходится собирать всех потомков в набор узлов. Это вынуждает нас пользоваться расширением `node-set()`. К тому же это означает, что функция `id()` не поможет нам найти супруга, так как она работает только относительно порядка документа. Однако узлы, представляющие потомков, – это копии исходных узлов и потому не принадлежат тому же документу. Из-за этого поиск элементов, относящихся к супругам, превращается в громоздкий обход переменной, содержащей все элементы person. Сравните с решением на языке XQuery:

```
define function descrip (element $e) returns element
{
  let $kids := $e/* union $e/@spouse=>person/*
  let $mstatus :=  if ($e[@spouse]) then "Yes" else "No"
  return
    <person married={ $mstatus } nkids={ count($kids) }>{ $e/@name/text( ) }
    </person>

define function descendants (element $e)
{
  if (empty($e/* union $e/@spouse=>person/*))
  then $e
  else $e union descendants($e/* union $e/@spouse=>person/*)
}

descrip(descendants(//person[@name = "Joe"])) sortby(@nkids descending, .)
```

Обсуждение

XSLT 1.0

В отличие от других примеров в настоящей книге, эта глава представляет собой что-то вроде шведского стола. Опрашивать XML можно столь разными способами, что придумать какие-то общие рецепты сложно. Консорциум W3C проделал заслуживающую уважения работу по классификации запросов, возникающих в разных сферах. Демонстрация того, как эти запросы можно реализовать на XSLT, позволит вам самостоятельно найти решения аналогичных задач.

Из-за нехватки места я не включил в эту главу решения на языке XQuery. Тем не менее, сопоставить оба подхода было бы поучительно, поэтому я призываю вас ознакомиться с документом *W3C Query Use Case*.

Комментировать каждый из приведенных выше примеров было бы непрактично. Но большинство читателей, владеющих основами XSLT, скорее всего, поймут решения без особого труда. Для некоторых решений возможны более удачные альтернативы. На мой выбор существенное влияние оказали решения на языке XQuery, предложенные в оригинальном документе W3C. Однако я все же пытался применять различные конструкции XSLT, иногда отдавая предпочтение итеративному стилю (`xsl:for-each`), а иногда – декларативному с использованием образцов и команды `xsl:apply-templates`.

XSLT 2.0

Эта глава получилась довольно объемной, поэтому я не дублировал все решения на XSLT 2.0. Однако приведу несколько задач, на примере которых демонстрируется, как можно улучшить решение за счет использования средств XPath 2.0 и XSLT 2.0:

- Пользуйтесь командой `for-each-group` или функцией `distinct-values()` для устранения дубликатов.

Вопрос 4. Для всех авторов в файле bib.xml вывести имя автора и названия написанных им книг, поместив все это внутрь элемента result:

```
<xsl:for-each-group select="//author" group-by=".">
  <result>
    <xsl:copy-of select="."/>
    <xsl:for-each select="/bib/book[author eq current-grouping-key( )]">
       <xsl:copy-of select="title"/>
    </xsl:for-each>
  </result>
</xsl:for-each-group>

<!-- С использованием distinct-values( ) -->

<xsl:for-each select="distinct-values(//author)">
  <result>
    <xsl:copy-of select="."/>
    <xsl:for-each select="/bib/book[author eq .]">
       <xsl:copy-of select="title"/>
    </xsl:for-each>
  </result>
</xsl:for-each>
```

- Избегайте копирования узлов, пользуйтесь вместо этого командой `xsl:sequence`.

Вопрос 8. В документе books.xml найти названия всех разделов и глав, в которых встречается слово "XML", независимо от уровня вложенности:

```
<results>
  <xsl:sequence select="(//chapter | //section)/
title)[contains(.,'XML')]"/>
</results>
```

- Для упрощения запросов пользуйтесь функциями и последовательностями.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ckbk="http://www.ora.com/XSLTCkbk">

<xsl:key name="person-key" match="person" use="@name"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:variable name="everyone" select="//person" as="item( )*"/>

<xsl:template match="census">
  <result>
    <xsl:apply-templates select="//person[@name='Joe']"/>
  </result>
</xsl:template>

<xsl:template match="person">

  <!-- Нет необходимости преобразовывать набор узлов.
  Воспользуйтесь функцией descendants напрямую -->
  <xsl:for-each select="ckbk:descendants(.,/)[current( ) != .]">
    <xsl:sort select="count(./* | $everyone[@name = current( )/@spouse]/*)"
    order="descending" data-type="number"/>
    <xsl:sort select="@name"/>
    <xsl:variable name="mstatus" select="if (@spouse) then 'Yes' else 'No'"/>
    <person married="{$mstatus}"
           nkids="{count(./* | key('person-key', @spouse)/*)}">
           <xsl:value-of select="@name"/>
    </person>
  </xsl:for-each>
</xsl:template>

<!-- Обратите внимание, насколько эта функция проще шаблона из решения
для XSLT 1.0. Мы передаем doc, поскольку внутри функции документ
неизвестен. -->
<xsl:function  name="ckbk:descendants">
```

```
  <xsl:param name="nodes" as="item( )*"/>
  <xsl:param name="doc"/>
  <xsl:sequence select="$nodes, for $person in $nodes return ckbk:descendants(
    ($person/person, key('person-key', $person/@spouse,$doc)/person), $doc)"/>
</xsl:function>

</xsl:stylesheet>
```

См. также

Эван Ленц также исследовал тему использования XSLT вместо XQuery (*http://xmlportfolio.com/xquery.html*).

Глава 10. Преобразование XML в HTML

Я был удивлен тем,
что люди так терпеливо
пишут HTML-код.
Тим Бернерс-Ли

10.0. Введение

Рискну высказать предположение, что по меньшей мере 60 процентов HTML-кода, опубликованного сегодня в Интернете, сгенерировано программно. Не потому, что HTML-код так трудно писать вручную, как намекает Тим Бернерс-Ли в цитате, вынесенной в эпиграф к этой главе (трудно, конечно, но теперь у нас есть мощные HTML-редакторы), а потому что с помощью динамической генерации можно достичь куда большего.

Существует множество открытых и патентованных технологий порождения HTML-контента из данных, хранящихся в той или иной форме. Но, если данные уже представлены в формате XML, то XSLT – один из важнейших инструментов преобразования, о котором должен знать каждый, кто занимается публикацией в Web.

Есть три основных способа использования XSLT для генерации HTML.

Во-первых, с помощью XSLT можно преобразовать XML в HTML и статически сохранить результат на Web-сервере или на жестком диске для доставки браузеру пользователя. Это неплохой способ тестирования преобразований.

Во-вторых, XSLT может служить основой серверного сценария, который извлекает XML-разметку из плоских файлов или баз данных и динамически преобразует ее на стороне Web-сервера по запросу от клиентского браузера. Это решение применяется, если исходные данные часто изменяются. Иногда используют и смешанный подход, то есть HTML-разметка генерируется по запросу, а затем кэшируется, чтобы не выполнять повторное преобразование не изменившихся данных. Если вы собираетесь применять XSLT на сервере, то определенно должны познакомиться с системой Apache Cocoon (*http://cocoon.apache.org*).

В-третьих, можно использовать таблицы стилей на стороне клиента, если браузер поддерживает обработку XSLT. В настоящее время такую поддержку предоставляют последние версии Microsoft Internet Explorer (версия 6.0 и более поздние), Netscape Navigator (6.1 и более поздние), Mozilla, Firefox и Apple Macintosh Safari (Tiger). Для более ранних версий IE требуется установить MSXML 3.0 в режиме замены. Кроме того, обработку XSLT на стороне клиента с выводом результатов производят программы XSmiles (*http://www.x-smiles.org/*)

и Antenna House XSL Formatter (*http://www.antennahouse.com/*). XSmiles может работать с данными в самых разных форматах, в том числе SVG и XSL-FO, хотя поддержка HTML оставляет желать лучшего. Antenna House XSL Formatter умеет обрабатывать XSL-FO. Поскольку ситуация меняется очень быстро, рекомендую справиться с последней версией онлайновой документации по вашему любимому браузеру или надстройке.

10.1. Использование XSLT в качестве языка стилизации

Задача

Вы хотите, чтобы браузер динамически стилизовал XML-документ, представив его в формате HTML.

Решение

Вполне достаточно решения на XSLT 1.0. Ниже приведен пример преобразования фрагмента документа в формате DocBook в HTML с использованием таблицы стилей XSLT. В качестве исходного документа взят кусок этой главы.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="application/xml" href="chapter.xsl"?>
<chapter label="8">
  <chapterinfo>
    <author>
      <surname>Мангано</surname>
      <firstname>Сэл</firstname>
    </author>
    <copyright>
      <year>2002</year>
      <holder>O'Рейли</holder>
    </copyright>
  </chapterinfo>
  <title>Преобразование XML в HTML</title>
  <epigraph>
    <para>Я был удивлен тем, что люди так терпеливо пишут HTML-код.</para>
    <attribution>Тим Бернерс-Ли</attribution>
  </epigraph>
  <sect1>
    <title>Использование XSLT в качестве языка стилизации</title>
    <sect2>
      <title>Задача</title>
      <para>Вы хотите, чтобы браузер динамически стилизовал
       XML-документ, представив его в формате HTML.</para>
```

```
    </sect2>
    <sect2>
      <title>Решение</title>
      <para>Ниже приведен пример преобразования фрагмента документа
      в формате DocBook в HTML с использованием таблицы стилей XSLT.
      В качестве исходного документа взят кусок этой главы.</para>
    </sect2>
    <sect2>
      <title>Обсуждение</title>
      <para>DocBook — пример документо-центрической DTD-схемы, которая
      позволяет создавать и сохранять документ в виде логической
      структуры, не зависящей от способа представления. Прелесть создания
      документов (особенно технических) в такой форме заключается
      в том, что с помощью XSLT единственное описание содержимого можно
      преобразовать в самые разные форматы, включая HTML, PDF, Microsoft
      Help и страницы руководства Unix. Хотя в этом рецепте используется
      схема DocBook, та же техника применима и к другим схемам документов —
      как общепризнанным, так и вашего изобретения. </para>
    </sect2>
  </sect1>
</chapter>
```

Обратите внимание, что вторая строка документа содержит команду обработки `xml-stylesheet`. Это указание браузеру применить следующую далее таблицу стилей к XML-разметке и вывести результат преобразования, а не исходный XML-код. (Напомню, что эта команда работает только в последних версиях браузеров.)

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

  <xsl:template match="/">
  <html>
    <head>
      <xsl:apply-templates mode="head"/>
    </head>
    <!-- Возможно, вы предпочтете использовать таблицу CSS,
    а не зашивать стили в код, как сделал я. -->
     <body style="margin-left:100;margin-right:100;margin-top:50;
        margin-bottom:50">
    <xsl:apply-templates/>
      <xsl:apply-templates select="chapter/chapterinfo/*" mode="copyright"/>
    </body>
  </html>
```

```
  </xsl:template>

  <!-- Head -->

  <xsl:template match="chapter" mode="head">
   <xsl:apply-templates select="chapterinfo" mode="head" />
   <xsl:apply-templates select="title" mode="head" />
  </xsl:template>

  <xsl:template match="chapter/title" mode="head">
   <title><xsl:value-of select="."/></title>
  </xsl:template>

  <xsl:template match="author" mode="head">
   <meta name="author" content="{concat(firstname,' ', surname)}"/>
  </xsl:template>

  <xsl:template match="copyright" mode="head">
    <meta name="copyright" content="{concat(holder,' ',year)}"/>
  </xsl:template>

  <xsl:template match="text(  )" mode="head"/>

<!-- Body -->

  <xsl:template match="chapter">
  <div align="right" style="font-size : 48pt; font-family: Times serif; font-
weight : bold; padding-bottom:10; color:red"><xsl:value-of select="@label"/>
</div>
  <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="chapter/title">
  <div align="right" style="font-size : 24pt; font-family: Times serif;
padding-bottom:150; color:red"><xsl:value-of select="."/></div>
  </xsl:template>

  <xsl:template match="epigraph/para">
  <div align="right" style="font-size : 10pt; font-family: Times serif; font-
style : italic; padding-top:4; padding-bottom:4"><xsl:value-of select="."/>
</div>
  </xsl:template>

  <xsl:template match="epigraph/attribution">
```

```
  <div align="right" style="font-size : 10pt; font-family: Times serif;
padding-top:4; padding-bottom:4"><xsl:value-of select="."/></div>
  </xsl:template>

  <xsl:template match="sect1">
  <h1 style="font-size : 18pt; font-family: Times serif; font-weight : bold">
    <xsl:value-of select="title"/>
  </h1>
  <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="sect2">
  <h2 style="font-size : 14pt; font-family: Times serif; font-weight : bold">
    <xsl:value-of select="title"/>
  </h2>
  <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="para">
   <p style="font-size : 12pt; font-family: Times serif">
  <xsl:value-of select="."/>
   </p>
  </xsl:template>

  <xsl:template match="text(  )"/>

<xsl:template match="copyright" mode="copyright">
  <div style="font-size : 10pt; font-family: Times serif; padding-top :
100">
    <xsl:text>Copyright </xsl:text>
    <xsl:value-of select="holder"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="year"/>
    <xsl:text>. Все права зарезервированы.</xsl:text>
  </div>
</xsl:template>

<xsl:template match="*" mode="copyright"/>

</xsl:stylesheet>
```

В результате браузер выведет следующий HTML-код:

```
<html>
  <head>
```

```
  <meta name="author" content="Сэл Мангано">
  <meta name="copyright" content="O'Рейли 2002">
  <title>XML to HTML</title>
  </head>
    <body  style="margin-left:100;margin-right:100;margin-top:50;margin-
bottom:50">
    <div align="right" style="font-size : 48pt; font-family: Times serif;
font-weight : bold; padding-bottom:10; color:red">8</div>
    <div align="right" style="font-size : 24pt; font-family: Times serif;
padding-bottom:150; color:red">Преобразование XML в HTML</div>
    <div align="right" style="font-size : 10pt; font-family: Times serif;
font-style : italic; padding-top:4; padding-bottom:4">Я был удивлен тем,
что люди так терпеливо пишут HTML-код.</div>
    <div align="right" style="font-size : 10pt; font-family: Times serif;
padding-top:4; padding-bottom:4">Тим Бернерс-Ли</div>
    <h1 style="font-size : 18pt; font-family: Times serif; font-weight :
bold">Использование XSLT в качестве языка стилизации</h1>
    <h2 style="font-size : 14pt; font-family: Times serif; font-weight :
bold">Задача</h2>
  <p style="font-size : 12pt; font-family: Times serif">Вы хотите, чтобы
браузер динамически стилизовал XML-документ, представив его в формате
HTML.</p>
    <h2 style="font-size : 14pt; font-family: Times serif; font-weight :
bold">Решение</h2>
  <p style="font-size : 12pt; font-family: Times serif">Ниже приведен
пример преобразования фрагмента документа в формате DocBook в HTML с
использованием таблицы стилей XSLT. В качестве исходного документа взят
кусок этой главы. </p>
    <h2 style="font-size : 14pt; font-family: Times serif; font-weight :
bold">Обсуждение</h2>
  <p style="font-size : 12pt; font-family: Times serif">DocBook —
пример документо- центрической DTD-схемы, которая позволяет создавать
и сохранять документ в виде логической структуры, не зависящей от способа
представления. Прелесть создания документов (особенно технических)
в такой форме заключается в том, что с помощью XSLT единственное
описание содержимого можно преобразовать в самые разные форматы,
включая HTML, PDF, Microsoft Help и страницы руководства Unix. Хотя
в этом рецепте используется схема DocBook, та же техника применима
и к другим схемам документов — как общепризнанным, так и вашего
изобретения.</p>
    <div style="font-size : 10pt; font-family: Times serif; padding-top :
100">Copyright O'Рейли 2002. Все права зарезервированы.</div>
    </body>
</html>
```

Обсуждение

DocBook – пример документо-центрической DTD-схемы, которая позволяет создавать и сохранять документ в виде логической структуры, не зависящей от способа представления. Прелесть создания документов (особенно технических) в такой форме заключается в том, что с помощью XSLT единственное описание содержимого можно преобразовать в самые разные форматы, включая HTML, PDF, Microsoft Help и страницы руководства Unix. Хотя в этом рецепте используется схема DocBook, та же техника применима и к другим схемам документов – как общепризнанным, так и вашего изобретения.

Поскольку мы использовали лишь малое подмножество DocBook DTD, то удобно оказалось создать простую монолитную таблицу стилей. Однако в решении промышленного качества надо было бы разнести обработку различных элементов DocBook по отдельным модулям в разных таблицах стилей и применять шаблоны в различных режимах.

Очевидный недостаток приведенного решения состоит в том, что информация о стилях «зашита» непосредственно в таблицу. Поскольку почти все современные браузеры поддерживают тот или иной уровень спецификации каскадных таблиц стилей (CSS), то было бы правильнее оставить XSLT только структурное преобразование, а стилизацию поручить CSS.

Базовая структура таблицы стилей сохраняется, но зашитые атрибуты `style` мы заменим атрибутами, определенными в файле `style.css`, ссылка на который добавлена в элемент `head`:

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

  <xsl:template match="/">
  <html>
     <head>
    <xsl:apply-templates mode="head"/>
    <link href="style.css" rel="stylesheet" type="text/css"/>
     </head>
     <body>
    <xsl:apply-templates/>
    <xsl:apply-templates select="chapter/chapterinfo/*" mode="copyright"/>
     </body>
  </html>

  </xsl:template>

  <!-- Head -->

  <xsl:template match="chapter" mode="head">
```

```
  <xsl:apply-templates select="chapterinfo" mode="head" />
     <xsl:apply-templates select="title" mode="head" />
  </xsl:template>

  <xsl:template match="chapter/title" mode="head">
     <title><xsl:value-of select="."/></title>
  </xsl:template>

  <xsl:template match="author" mode="head">
     <meta name="author" content="{concat(firstname,' ', surname)}"/>
  </xsl:template>

  <xsl:template match="copyright" mode="head">
     <meta name="copyright" content="{concat(holder,' ',year)}"/>
  </xsl:template>

  <xsl:template match="text(  )" mode="head"/>

<!-- Body -->

  <xsl:template match="chapter">
   <div class="chapter"><xsl:value-of select="@label"/></div>
   <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="chapter/title">
   <div class="title"><xsl:value-of select="."/></div>
  </xsl:template>

  <xsl:template match="epigraph/para">
   <div class="epigraph"><xsl:value-of select="."/></div>
  </xsl:template>

  <xsl:template match="epigraph/attribution">
  <div class="epigraph-attribution"><xsl:value-of select="."/></div>
  </xsl:template>

  <xsl:template match="sect1">
  <h1><xsl:value-of select="title"/></h1>
  <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="sect2">
   <h2><xsl:value-of select="title"/></h2>
```

```
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="para">
    <p class="para"><xsl:value-of select="."/></p>
  </xsl:template>

  <xsl:template match="text()"/>

<xsl:template match="copyright" mode="copyright">
  <div class="copyright">
    <xsl:text>Copyright </xsl:text>
    <xsl:value-of select="holder"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="year"/>
    <xsl:text>. Все права зарезервированы.</xsl:text>
  </div>
</xsl:template>

<xsl:template match="*" mode="copyright"/>

</xsl:stylesheet>
```

Таблица CSS-стилей – это плоский ASCII-файл (*style.css*), следующий простому набору соглашений:

```
body
{
    margin-left:100;
    margin-right:100;
    margin-top:50;
    margin-bottom:50;
    font-family: Times serif;
    font-size : 12pt;
    color:black;
}

div.chapter
{
    text-align:right;
    font-size : 48pt;
    font-weight: bold;
    padding-bottom:10;
    color:red;
}

div.title
```

```
{
    font-size : 24pt;
    font-family: Times serif;
    padding-bottom:150;
    color:red"
}

div.epigraph
{
    font-style:: italic;
}

div.epigraph, div.epigraph-attribution
{
    text-align: right;
    font-size : 10pt;
    padding-top: 4;
    padding-bottom: 4;
}

h1
{
    font-size : 18pt;
    font-weight : bold;
}

h2
{
    font-size : 14pt;
    font-weight : bold";
}
```

Если вы не знакомы с CSS, то обратитесь к превосходному учебному руководству по адресу *http://www.w3schools.com/css/default.asp*. Рекомендую также книгу Eric Meyer *Cascading Style Sheets: The Definitive Guide*, Second Edition, O'Reilly, 2004[1]. Есть также много инструментов для создания CSS-таблиц. Лично мне нравится программа Macromedia Dreamweaver MX, но кто-то скажет, что она немного тяжеловата, если нужен всего лишь простой редактор CSS.

См. также

Лучший источник информации о формате DocBook – сайт *http://www.docbook.org/*. Норманн Уолш (Norman Walsh) разработал набор общедоступных таблиц стилей

[1] Эрик Мейер «Каскадные таблицы стилей. Подробное руководство», издательство Символ-Плюс, 2006.

для преобразования из DocBook в различные форматы публикации. Эти таблицы находятся по адресу *http://docbook.sourceforge.net/projects/xsl/*.

В рецепте 16.2 описывается несколько приемов создания более модульных и расширяемых таблиц стилей.

10.2. Создание документов, связанных гиперссылками

Задача

Требуется преобразовать XML-документ в набор HTML-файлов, связанных гиперссылками.

Решение

Обычно, преобразуя XML в HTML, выполняют два или более проходов по XML-файлу, чтобы создать индексные страницы и содержимое. Индексные страницы содержат ссылки на содержимое. (Для этого достаточно XSLT 1.0.) В следующем решении из файла *SalesBySalesPerson.xml* (см. главу 2) генерируется индекс и сводные страницы.

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:saxon="http://icl.com/saxon"
 extension-element-prefixes="saxon">

<xsl:output method="html"/>

<xsl:template match="/">
    <xsl:apply-templates select="*" mode="index"/>
    <xsl:apply-templates select="*" mode="content"/>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--        Создать index.html  (mode = "index")       -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<xsl:template match="salesBySalesperson" mode="index">
  <saxon:output href="index.html">
  <html>
    <head>
    <title>Индекс файла продаж в разрезе продавцов</title>
    </head>

    <body bgcolor="#FFFFFF" text="#000000">
    <h1>Продажи в разрезе продавцов</h1>
```

```
    <xsl:apply-templates mode="index"/>
    </body>
  </html>
  </saxon:output>
</xsl:template>

<xsl:template match="salesperson" mode="index">
  <h2>
  <a href="{concat(@name,'.html')}">
    <xsl:value-of select="@name"/>
  </a>
  </h2>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--      Создать @name.html  (mode = "content")       -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->

<xsl:template match="salesperson" mode="content">
  <saxon:output href="{@name}.html">
    <html>
      <head>
        <title><xsl:value-of select="@name"/> Sales</title>
      </head>

      <body bgcolor="#FFFFFF" text="#000000">
        <h1><xsl:value-of select="@name"/> Sales</h1>
        <ol>
          <xsl:apply-templates mode="content"/>
        </ol>
      </body>
    </html>
  </saxon:output>
</xsl:template>

<xsl:template match="product" mode="content">
  <li><xsl:value-of
    select="@sku"/>      $<xsl:value-
of select="@totalSales"/></li>
</xsl:template>

</xsl:stylesheet>
```

Обратите внимание, как с помощью режимов мы отделяем преобразование элементов XML в индекс от преобразования информационного содержимого

в HTML-страницы для каждого продавца. Режимы используются для преобразования в HTML очень часто, так как позволяют с помощью единственной таблицы стилей построить разные представления одного и того же документа.

Эта таблица спроектирована для пакетной обработки. Ее можно параметризовать, указав, какой документ создавать. Заодно параметр устранил бы необходимость в использовании нестандартного расширения `saxon:output` (которое было бы не нужно при использовании XSLT 2.0).

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

<xsl:output method="html"/>
<!-- Используется для задания выходного документа -->
<!-- INDEX : создать индексную страницу -->
<!-- Имя продавца : создать страницу для этого продавца -->
<xsl:param name="which" select="'INDEX'"/>

<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="$which='INDEX'">
       <xsl:apply-templates select="*" mode="index"/>
    </xsl:when>
    <xsl:otherwise>
       <xsl:apply-templates select="*/salesperson[@name = $which]"
                            mode="content"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--       Создать index.html  (mode = "index")        -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<xsl:template match="salesBySalesperson" mode="index">
    <!-- saxon:output удалено. Остальное не изменилось. -->
</xsl:template>

<!-- ... -->

<xsl:template match="salesperson" mode="content">
    <!-- saxon:output удалено. Остальное не изменилось. -->
</xsl:template>

<!-- ... -->

</xsl:stylesheet>
```

Этот подход станет надежнее, если в качестве параметра передавать не имя, а идентификатор продавца.

Обсуждение

В этом решении не создается ничего особенно впечатляющего, просто иллюстрируется механизм генерации связанного HTML-контента. Для создания всех Web-страниц одной таблицей стилей пришлось воспользоваться нестандартным для XSLT 1.0 элементом (`saxon:output`). Аналогичные расширения есть и во многих других процессорах, а в XSLT 2.0 появилась команда `xsl:result-document`.

Эта таблица стилей порождает относительные ссылки, что в большинстве случаев и требуется. Если же нужны абсолютные ссылки, то лучше не «зашивать» в код URL, а передать его в виде параметра:

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

<xsl:output method="html"/>

<xsl:param name="URL" select="http://www.mycompany.com/"/>

<!-- опущено ... -->

<xsl:template match="salesperson" mode="index">
  <h2>
    <a href="{$URL}{@name}.html'">
       <xsl:value-of select="@name"/>
    </a>
  </h2>
</xsl:template>
```

См. также

Дополнительную информацию о выводе нескольких документов см. в рецепте 6.6.

Показанное выше преобразование генерирует контент безо всяких эстетических излишеств. О том, как сделать результат приятнее для взгляда, см. рецепт 10.4.

10.3. Создание HTML-таблиц

Задача

Требуется преобразовать XML-документ в HTML-таблицы.

Решение

Таблицы часто создаются в два этапа. Сначала генерируется табличная разметка верхнего уровня, а затем для создания строк и ячеек применяются шаблоны.

Показанное ниже решение – это частичная модификация таблицы стилей из рецепта 10.2. Изменения выделены полужирным шрифтом:

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:param name="URL"/>

<xsl:template match="/">
  <xsl:apply-templates select="*" mode="index"/>
  <xsl:apply-templates select="*" mode="content"/>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--      Создать index.html  (mode = "index")         -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<xsl:template match="salesBySalesperson" mode="index">
  <!-- Нестандартный элемент xsl:document, реализованный в saxon! -->
  <xsl:document href="index.html">
  <html>
  <head>
    <title>Продажи в разрезе продавцов</title>
  </head>

  <body bgcolor="#FFFFFF" text="#000000">
    <h1>Продажи в разрезе продавцов</h1>
    <xsl:apply-templates mode="index"/>
  </body>
  </html>
  </xsl:document>
</xsl:template>

<xsl:template match="salesperson" mode="index">
  <h2>
  <a href="{concat($URL,@name,'.html')}">
    <xsl:value-of select="@name"/>
  </a>
  </h2>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--      Создать @name.html  (mode = "content")       -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->

<xsl:template match="salesperson" mode="content">
```

```
  <xsl:document href="{concat(@name,'.html')}">
   <html>
  <head>
     <title><xsl:value select="@name"/></title>
  </head>

  <body bgcolor="#FFFFFF" text="#000000">
  <h1><xsl:value-of select="@name"/> Sales</h1>
  <table border="1" cellpadding="3">
    <tbody >
      <tr>
        <th>Позиция</th>
        <th>Объем продаж (в US $)</th>
      </tr>
      <xsl:apply-templates mode="content"/>
    </tbody>
  </table>
  <h2><a href="{concat($URL,'index.html')}">Home</a></h2>
  </body>
   </html>
  </xsl:document>
</xsl:template>

<xsl:template match="product" mode="content">
   <tr>
  <td><xsl:value-of select="@sku"/></td>
  <td><xsl:value-of select="@totalSales"/></td>
   </tr>

</xsl:template>

</xsl:stylesheet>
```

Обсуждение

XSLT 1.0

При создании таблиц данные часто приходится группировать по некоторому критерию. Трудность этой задачи зависит от того, сгруппированы ли данные изначально и насколько определенным является критерий. Предположим, к примеру, что требуется сгруппировать данные о продажах по регионам:

```
<?xml version="1.0" encoding="UTF-8"?>
<sales>
    <product sku="10000" sales="90000.00" region="NE"/>
    <product sku="10000" sales="10000.00" region="NW"/>
```

```
    <product sku="10000" sales="55000.00" region="SE"/>
    <product sku="10000" sales="32000.00" region="SW"/>
    <product sku="10000" sales="95000.00" region="NC"/>
    <product sku="10000" sales="88000.00" region="SC"/>
    <product sku="20000" sales="77000.00" region="NE"/>
    <product sku="20000" sales="11100.00" region="NW"/>
    <product sku="20000" sales="33210.00" region="SE"/>
    <product sku="20000" sales="78000.00" region="SW"/>
    <product sku="20000" sales="105000.00" region="NC"/>
    <product sku="20000" sales="12300.00" region="SC"/>
    <product sku="30000" sales="1000.00" region="NE"/>
    <product sku="30000" sales="5100.00" region="NW"/>
    <product sku="30000" sales="3210.00" region="SE"/>
    <product sku="30000" sales="8000.00" region="SW"/>
    <product sku="30000" sales="5000.00" region="NC"/>
    <product sku="30000" sales="11300.00" region="SC"/>
</sales>
```

Здесь заранее известно, что есть шесть регионов. Поэтому задачу группировки можно решить путем явного отбора:

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

<xsl:template match="sales">
  <html>
    <head>
    <title>Продажи в разрезе регионов</title>
   </head>
   <body>
    <h1>Продажи в разрезе регионов</h1>
       <table border="1" cellpadding="3">
           <tbody>
           <tr>
              <th>Позиция</th>
              <th>Продажи</th>
           </tr>
           <xsl:call-template name="group-region">
              <xsl:with-param name="region" select=" 'NE' "/>
              <xsl:with-param name="title" select="'North East Sales'"/>
           </xsl:call-template>
           <xsl:call-template name="group-region">
              <xsl:with-param name="region" select=" 'NW' "/>
              <xsl:with-param name="title" select="'North West Sales'"/>
           </xsl:call-template>
```

```
            <xsl:call-template name="group-region">
               <xsl:with-param name="region" select=" 'NC' "/>
               <xsl:with-param name="title" select="'North Central Sales'"/>
            </xsl:call-template>
            <xsl:call-template name="group-region">
               <xsl:with-param name="region" select=" 'SE' "/>
               <xsl:with-param name="title" select="'South East Sales'"/>
            </xsl:call-template>
            <xsl:call-template name="group-region">
               <xsl:with-param name="region" select=" 'SC' "/>
               <xsl:with-param name="title" select="'South Central Sales'"/>
            </xsl:call-template>
            <xsl:call-template name="group-region">
               <xsl:with-param name="region" select=" 'SW' "/>
               <xsl:with-param name="title" select="'South West Sales'"/>
            </xsl:call-template>
            </tbody>
        </table>
    </body>
   </html>
</xsl:template>

<xsl:template name="group-region">
  <xsl:param name="region"/>
  <xsl:param name="title"/>
  <xsl:variable name="products" select="product[@region = $region]" />
  <tr>
    <th colspan="2"><xsl:value-of select="$title" /></th>
  </tr>
  <xsl:apply-templates select="$products"/>
  <tr style="font-weight:bold">
    <td >Total</td>
    <td align="right">
     <xsl:value-of
           select="format-number(sum($products/@sales), '#.00')"/>
    </td>
   </tr>
</xsl:template>

<xsl:template match="product">
  <tr>
   <td><xsl:value-of select="@sku"/></td>
   <td align="right"><xsl:value-of select="@sales"/></td>
  </tr>
```

```
</xsl:template>

</xsl:stylesheet>
```

Если «зашивать» названия групп в код вам не нравится, примените таблично-управляемый подход:

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

    <sales:region code=__QUOT?QUOT__ name="North East"/>
    <sales:region code="NC" name="North Central"/>
    <sales:region code="NW" name="North West"/>
    <sales:region code="SE" name="South East"/>
    <sales:region code="SC" name="South Central"/>
    <sales:region code="SW" name="South West"/>

<xsl:variable name="products" select="/sales/product"/>

<xsl:output method="html"/>

<xsl:template match="sales">
  <html>
    <head>
    <title>Продажи в разрезе регионов</title>
   </head>
   <body>
   <h1>Продажи в разрезе регионов</h1>
   <table border="1" cellpadding="3">
    <tbody>
       <tr>
            <th>Позиция</th>
            <th>Продажи</th>
       </tr>
       <xsl:for-each select="document('')/*/sales:region">
            <tr >
               <th colspan="2"><xsl:value-of select="@name"/> Продажи</th>
            </tr>
            <xsl:call-template name="group-region">
               <xsl:with-param name="region" select="@code"/>
            </xsl:call-template>
       </xsl:for-each>
    </tbody>
   </table>
```

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

<xsl:template name="group-region">
  <xsl:param name="region"/>
  <xsl:apply-templates select="$products[@region=$region]"/>
  <tr style="font-weight:bold">
    <td>Итого</td>
    <td align="right"><xsl:value-of
      select="format-number(sum($products[@region=$region]/@sales),'#.00')"/>
    </td>
  </tr>
</xsl:template>

<xsl:template match="product">
  <tr>
    <td><xsl:value-of select="@sku"/></td>
    <td align="right"><xsl:value-of select="@sales"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

Разумеется, не все задачи группировки настолько просты. Пусть, например, нужно спроектировать таблицу стилей, которая будет использоваться несколькими компаниями или подразделениями одной компании, у которых собственные соглашения об именовании регионов продаж. К этой задаче можно подойти с разных сторон, но одним из самых эффективных является метод группировки Мюнха (названный в честь придумавшего его Стива Мюнха из корпорации Oracle), в котором используется комбинация ключей и идентификаторов:

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

<xsl:output method="html"/>

<xsl:key name="region-key" match="product" use="@region"/>

  <xsl:template match="sales">
  <html>
    <head>
     <title>Продажи в разрезе регионов</title>
    </head>
    <body>
```

```
      <h1>Продажи в разрезе регионов</h1>
      <table border="1" cellpadding="3">
     <tbody>
        <tr>
        <th>Позиция</th>
        <th>Sales</th>
     </tr>
     <xsl:variable name="unique-regions"
        select="/sales
                /product[generate-id(.) =
                          generate-id(key('region-key',@region))]
                /@region"/>
     <xsl:for-each select="$unique-regions">
        <tr>
          <th colspan="2"><xsl:value-of select="."/> Sales</th>
        </tr>
        <xsl:call-template name="group-region">
          <xsl:with-param name="region" select="."/>
        </xsl:call-template>
       </xsl:for-each>
     </tbody>
    </table>
   </body>
  </html>
  </xsl:template>

  <xsl:template name="group-region">
  <xsl:param name="region"/>
    <xsl:apply-templates select="key('region-key', @region)"/>
    <tr style="font-weight:bold">
    <td >Total</td>
    <td align="right"><xsl:value-of
      select="format-number(sum(key('region-key',  @region)/@sales),'#.00')"/>
    </td>
    </tr>
  </xsl:template>

  <xsl:template match="product">
  <tr>
    <td><xsl:value-of select="@sku"/></td>
    <td align="right"><xsl:value-of select="@sales"/></td>
  </tr>
  </xsl:template>

</xsl:stylesheet>
```

По своей структуре это решение аналогично таблично-управляемому. Основное различие в том, как определяется уникальный набор групп. В таблично-управляемом случае множество регионов кодируется вручную в элементах `sales:region`. А в методе Мюнха для определения значений групп используется ключ. Известно, что выражение `key('region-key',@region)` возвращает набор узлов, соответствующих всем товарным позициям в регионе `@region`. Известно также, что функция `generate-id` возвращает уникальный идентификатор первого элемента в переданном ей наборе узлов. Таким образом, выражение `[generate-id(.) = generate-id( key( 'region-key', @region))]` истинно только для первого элемента в каждой группе, и в данном случае это позволяет получить все уникальные регионы, составляющие группу. Наличие ключа заодно повышает эффективность работы других частей таблицы стилей, которые ссылаются на товар по региону.

XSLT 2.0

Если вы не начали читать прямо с этой главы, то знаете, что в XSLT 2.0 выполнять группировку лучше всего с помощью команды `xsl:for-each-group`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html"/>

<xsl:template match="sales">
  <html>
   <head>
  <title>Продажи в разрезе регионов</title>
   </head>
   <body>
  <h1>Продажи в разрезе регионов</h1>
  <table border="1" cellpadding="3">
    <tbody>
    <tr>
       <th>Позиция</th>
       <th>Продажи</th>
    </tr>
    <xsl:for-each-group select="product" group-by="@region">
       <tr>
        <th colspan="2"><xsl:value-of select="."/> Продажи</th>
      </tr>
      <xsl:apply-templates select="current-group()"/>
      <tr style="font-weight:bold">
         <td>Итого</td>
         <td align="right"><xsl:value-of
```

```
          select="format-number(sum(current-group()/@sales),'#.00')"/>
           </td>
         </tr>
        </xsl:for-each-group>
      </tbody>
    </table>
  </body>
 </html>
</xsl:template>

<xsl:template match="product">
  <tr>
    <td><xsl:value-of select="@sku"/></td>
    <td align="right"><xsl:value-of select="@sales"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

10.4. Создание фреймов

Задача

Требуется сгенерировать HTML-разметку, в которой содержимое организовано с помощью фреймов.

Решение

Как и в рецепте 10.2, воспользуемся режимами для выполнения нескольких проходов по XML-документу. Сначала создается контейнер фреймов, содержащий тег <FRAMESET>. Мы создадим всего два фрейма; меньший фрейм слева будет содержать имена продавцов в виде гиперссылок, загружающих содержимое в правый фрейм. Правый – основной – фрейм будет содержать данные о продажах для выбранного пользователем продавца. В примере генерируется также документ, отображаемый в основном фрейме при первой загрузке страницы.

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

<xsl:output method="html"/>

<xsl:param name="URL"/>

<xsl:template match="/">
  <xsl:apply-templates select="*" mode="frameset"/>
  <xsl:apply-templates select="*" mode="salespeople_frame"/>
```

```
  <xsl:apply-templates select="*" mode="sales_frames"/>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--  Создать контейнер для фреймов (mode ="frameset") -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<xsl:template match="salesBySalesperson" mode="frameset">
  <!-- Нестандартный элемент saxon xsl:document! -->
  <xsl:document href="index.html">
    <html>
      <head>
        <title>Данные о продавцах</title>
      </head>
      <frameset rows="100%" cols="25%, 75%" border="0">
        <frame name="salespeople" src="salespeople_frame.html" noresize=""/>
        <frame name="mainFrame" src="default_sales.html" noresize=""/>
      </frameset>
    <body bgcolor="#FFFFFF" text="#000000">
    </body>
  </html>
 </xsl:document>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!-- Создать salespeople_frame.html  (mode = "salespeople_frame") -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<xsl:template match="salesBySalesperson" mode="salespeople_frame">
  <!-- Нестандартный элемент saxon xsl:document! -->
  <xsl:document href="salespeople_frame.html">
    <html>
     <head>
       <title>Salespeople</title>
     </head>
     <body bgcolor="#FFFFFF" text="#000000">
       <table>
        <tbody>
          <xsl:apply-templates mode="index"/>
        </tbody>
      </table>
     </body>
    </html>
  </xsl:document>
</xsl:template>

<xsl:template match="salesperson" mode="index">
```

```
  <tr>
    <td>
      <a href="{concat(@name,'.html')}"
          target="mainFrame"><xsl:value-of select="@name"/></a>
    </td>
  </tr>
</xsl:template>

<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->
<!--        Создать @name.html  (mode = "content")     -->
<!-- =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  = -->

<xsl:template match="salesperson" mode="sales_frames">

  <xsl:document href="default_sales.html">
    <html>
     <head>
       <title>По умолчанию</title>
     </head>

     <body bgcolor="#FFFFFF" text="#000000">
     <h1><center>Продажи для одного продавца</center></h1>
      <br/>
    Щелкните по имени продавца в левом фрейме,
    чтобы загрузить данные о продажах для него.
     </body>
    </html>
  </xsl:document>

  <xsl:document href="{concat(@name,'.html')}">
    <html>
     <head>
       <title><xsl:value-of select="@name"/></title>
     </head>

     <body bgcolor="#FFFFFF" text="#000000">
     <h1><center>Продажи для одного продавца</center></h1>
      <h2><xsl:value-of select="@name"/></h2>
      <table border="1" cellpadding="3">
        <tbody >
          <tr>
            <th>Позиция</th>
            <th>Продажи (в US $)</th>
          </tr>
```

```
            <xsl:apply-templates mode="content"/>
          </tbody>
        </table>
      </body>
    </html>
  </xsl:document>
</xsl:template>

<xsl:template match="product" mode="content">
  <tr>
    <td><xsl:value-of select="@sku"/></td>
    <td align="right"><xsl:value-of select="@totalSales"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Фреймы полезны, когда нужно разбить страницу на логические разделы, однако применять фреймы с видимыми или перемещаемыми рамками уже несовременно. Приведенную выше таблицу стилей нельзя использовать на стороне клиента, поскольку она выводит несколько файлов: один для контейнера, другой для левого фрейма, в котором перечисляются имена продавцов, и по одному для каждого продавца.

Показанная таблица стилей при обработке одного XML-документа выводит заранее неизвестное число страниц. Но можно работать с фреймами, применяя XSLT-преобразования и на стороне клиента. Достаточно создать страницу-контейнер, а в каждый фрейм загрузить отдельный XML-документ с собственным преобразованием:

```
<html>
    <head>
       <title>Frameset</title>
    </head>
    <frameset rows="100%" cols="25%, 75%" border="0">
       <frame name="leftFrame" src="left.xml" noresize="">
       <frame name="mainFrame" src="main.xml" noresize="">
    </frameset>
    <body bgcolor="#FFFFFF" text="#000000"></body>
</html>
```

Загружать в разные фреймы различные XML-документы приходится потому, что в файле может быть не более одной команды `xml-stylesheet`. Однако значимое содержимое можно по-прежнему хранить в одной таблице стилей, воспользовавшись командой `xinclude` (*http://www.w3.org/TR/xinclude/*) и XSLT-шаблоном для ее обработки:

Файл *left.xml* выглядит так:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="application/xml" href="left.xsl"?>
<xi:include href="salesBySalesperson.xml"
            xmlns:xi="http://www.w3.org/2001/XInclude"/>
```

а файл *main.xml* – так:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="application/xml" href="main.xsl"?>
<xi:include href="salesBySalesperson.xml"
            xmlns:xi="http://www.w3.org/2001/XInclude"/>
```

Каждый из файлов *left.xsl* и *main.xsl* содержит шаблон, который обрабатывает элемент `xinclude` с помощью функции `document`:

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

<xsl:output method="html"/>

<xsl:template match="xi:include" xmlns:xi="http://www.w3.org/2001/XInclude">
    <xsl:for-each select="document(@href,.)">
           <xsl:apply-templates/>
    </xsl:for-each>
</xsl:template>

<xsl:template match="salesBySalesperson">
  <html>
   <head>
    <title>Продавцы</title>
   </head>
   <body bgcolor="#FFFFFF" text="#000000">
     <table>
      <tbody>
        <xsl:apply-templates/>
      </tbody>
    </table>
   </body>
  </html>
</xsl:template>

<xsl:template match="salesperson">
  <tr>
    <td>
```

```
      <a href="{concat('main.xml#',@name)}" target="mainFrame">
      <xsl:value-of select="@name"/></a>
    </td>
  </tr>
</xsl:template>

</xsl:stylesheet>

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

<xsl:template match="xi:include" xmlns:xi="http://www.w3.org/2001/XInclude">
    <xsl:for-each select="document(@href)">
       <xsl:apply-templates/>
    </xsl:for-each>
</xsl:template>

<xsl:template match="salesBySalesperson">
    <html>
     <head>
      <title>Продажи для одного продавца</title>
     </head>

     <body bgcolor="#FFFFFF" text="#000000">
     <xsl:apply-templates/>
     </body>
    </html>
</xsl:template>

<xsl:template match="salesperson">
  <h1><a name="{@name}"><center>Продажи для одного продавца</center></a></h1>
     <h2><xsl:value-of select="@name"/></h2>
     <table border="1" cellpadding="3">
        <tbody >
          <tr>
            <th>Позиция</th>
            <th>Продажи (в US $)</th>
          </tr>
          <xsl:apply-templates />
        </tbody>
      </table>
      <div style="padding-top:1000"/>
</xsl:template>
```

```
<xsl:template match="product">
    <tr>
       <td><xsl:value-of select="@sku"/></td>
       <td align="right"><xsl:value-of select="@totalSales"/></td>
    </tr>
</xsl:template>

</xsl:stylesheet>
```

Таблица стилей *main.xsl* генерирует именованные теги `<a>` для каждого продавца и оставляет между ними большие промежутки с помощью тега `<div style="padding-top:1000"/>`. В таблице *left.xsl* генерируются соответствующие ссылки. Тем самым имитируется, хотя и довольно грубо, загрузка отдельной страницы для каждого продавца.

См. также

В книге Джени Теннисон *XSLT and XPath on the Edge* (M&T, 2001) подробно обсуждаются XSLT-преобразования на стороне клиента с участием фреймов и показывается, как с помощью JavaScript можно достичь более впечатляющих результатов.

10.5. Создание таблиц стилей, управляемых данными

Задача

Требует сгенерировать HTML-разметку, стилизованную в соответствии с отображаемыми данными.

Решение

XSLT 1.0

Наборы атрибутов в XSLT – отличный механизм для инкапсуляции сложностей, сопутствующих стилизации, которая управляется данными. Рассмотрим следующий XML-документ, описывающий инвестиционный портфель:

```
<portfolio>
  <investment>
    <symbol>IBM</symbol>
    <current>72.70</current>
    <paid>65.00</paid>
    <qty>1000</qty>
  </investment>
  <investment>
    <symbol>JMAR</symbol>
```

```
    <current>1.90</current>
    <paid>5.10</paid>
    <qty>5000</qty>
  </investment>
  <investment>
    <symbol>DELL</symbol>
    <current>24.50</current>
    <paid>18.00</paid>
    <qty>100000</qty>
    </investment>
  <investment>
    <symbol>P</symbol>
    <current>57.33</current>
    <paid>63</paid>
    <qty>100</qty>
  </investment>
</portfolio>
```

Можно вывести этот портфель в виде таблицы, содержащей колонку, где прибыль показана черным цветом, а убытки – красным:

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

  <xsl:output method="html"/>

  <xsl:attribute-set name="gain-loss-font">
  <xsl:attribute name="color">
     <xsl:choose>
      <xsl:when test="(current - paid) * qty >= 0">black</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
     </xsl:choose>
  </xsl:attribute>
  </xsl:attribute-set>

<xsl:template match="portfolio">
  <html>
   <head>
    <title>Мой портфель</title>
   </head>

     <body bgcolor="#FFFFFF" text="#000000">
      <h1>Портфель</h1>
      <table border="1" cellpadding="2">
        <tbody>
          <tr>
```

```
            <th>Символ</th>
            <th>Текущая цена</th>
            <th>Цена на момент покупки</th>
            <th>Количество</th>
            <th>Прибыли/Убытки</th>
          </tr>
          <xsl:apply-templates/>
        </tbody>
      </table>
     </body>
    </html>
</xsl:template>

<xsl:template match="investment">
  <tr>
    <td><xsl:value-of select="symbol"/></td>
    <td><xsl:value-of select="current"/></td>
    <td><xsl:value-of select="paid"/></td>
    <td><xsl:value-of select="qty"/></td>
    <td>
     <font xsl:use-attribute-sets="gain-loss-font">
      <xsl:value-of
        select="format-number((current - paid) * qty, '#,##0.00')"/>
     </font>
    </td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

Если совместимость со старыми браузерами вас не волнует, то можно сделать разметку изящнее, воспользовавшись атрибутом `style` вместо элемента `font`:

```
<xsl:attribute-set name="gain-loss-color">
  <xsl:attribute name="style">color:<xsl:text/>
    <xsl:choose>
           <xsl:when test="(current - paid) * qty >= 0">black</xsl:when>
           <xsl:otherwise>red</xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
</xsl:attribute-set>

...

<xsl:template match="investment">
```

```
  <tr>
    <td><xsl:value-of select="symbol"/></td>
    <td><xsl:value-of select="current"/></td>
    <td><xsl:value-of select="paid"/></td>
    <td><xsl:value-of select="qty"/></td>
    <td xsl:use-attribute-sets="gain-loss-color">
       <xsl:value-of
           select="format-number((current - paid) * qty, '#,##0.00')"/>
    </td>
  </tr>
</xsl:template>
```

XSLT 2.0

За счет использования версии 2.0 можно, прежде всего, сократить код, применив выражения XPath вместо команды `xsl:choose`. Например, переписать фрагмент кода следующим образом:

```
<xsl:attribute-set name="gain-loss-font">
  <xsl:attribute name="color"
                 select="if ((current - paid) * qty ge 0)
                        then 'black' else 'red'"/>
</xsl:attribute-set>
```

Обсуждение

Как обычно, XSLT позволяет решить задачу несколькими способами. Можно было бы включить логику стилизации непосредственно в код генерации таблицы:

```
<xsl:template match="investment">
  <tr>
   <td><xsl:value-of select="symbol"/></td>
   <td><xsl:value-of select="current"/></td>
   <td><xsl:value-of select="paid"/></td>
   <td><xsl:value-of select="qty"/></td>
   <td>
    <font>
       <xsl:attribute name="color">
         <xsl:choose>
           <xsl:when test="(current - paid) * qty >= 0">black</xsl:when>
         <xsl:otherwise>red</xsl:otherwise>
         </xsl:choose>
       </xsl:attribute>
       <xsl:value-of
          select="format-number((current - paid) * qty, '#,##0.00')"/>
      </font>
   </td>
```

```
 </tr>
</xsl:template>
```

Если код определения цвета встроен таким образом, то проще понять, что делает программа, но при этом усложняется логика создания таблицы. В более сложном примере атрибуты `style` могли бы вычисляться для многих элементов. Когда построение структуры смешано со стилизацией, таблицу стилей становится труднее понимать и модифицировать.

Можно, однако, возразить, что «зашивание» ссылок на элементы в наборы атрибутов не дает использовать их повторно. Обычно эту проблему можно решить, вынеся логику вычисления атрибутов из самого набора атрибутов за счет использования шаблонов с разными режимами. Рассмотрим портфель, состоящий из инвестиций, прибыльность которых вычисляется по-разному:

```
<portfolio>

    <stock>
       <symbol>IBM</symbol>
       <current>72.70</current>
       <paid>65.00</paid>
       <qty>1000</qty>
    </stock>

    <stock>
       <symbol>JMAR</symbol>
       <current>1.90</current>
       <paid>5.10</paid>
       <qty>5000</qty>
    </stock>

    <stock>
       <symbol>DELL</symbol>
       <current>24.50</current>
       <paid>18.00</paid>
       <qty>100000</qty>
    </stock>

    <stock>
       <symbol>P</symbol>
       <current>57.33</current>
       <paid>63.00</paid>
       <qty>100</qty>
    </stock>

    <property>
```

```
        <address>123 Main St. Anytown NY</address>
        <paid>100000</paid>
        <appriasal>250000</appriasal>
    </property>

    <property>
        <address>13 Skunks Misery Dr. Stinksville NJ</address>
        <paid>200000</paid>
        <appriasal>50000</appriasal>
    </property>

</portfolio>
```

Можно избежать определения двух наборов атрибутов, выполняющих одну и ту же функцию, вынеся логику в шаблоны:

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

  <xsl:output method="html"/>

  <xsl:attribute-set name="gain-loss-font">
    <xsl:attribute name="color">
      <xsl:apply-templates select="." mode="gain-loss-font-color"/>
    </xsl:attribute>
  </xsl:attribute-set>

<xsl:template match="stock" mode="gain-loss-font-color">
    <xsl:choose>
      <xsl:when test="(current - paid) * qty >= 0">black</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="property" mode="gain-loss-font-color">
    <xsl:choose>
      <xsl:when test="appriasal - paid  >= 0">black</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
    </xsl:choose>
</xsl:template>

...
</xsl:stylesheet>
```

Возможно, вам не очень нравится вставлять число стилистические атрибуты – цвета, шрифты и т.п. – в XSLT-преобразование. Быть может, это вообще не ваша работа решать, как отображать прибыли и убытки; на то есть дизайнеры. В таком

случае снабдите элементы классами и перенесите решение о стилизации на уровень таблицы CSS-стилей:

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

  <xsl:output method="html"/>

  <xsl:attribute-set name="gain-loss">
    <xsl:attribute name="class">
       <xsl:apply-templates select="." mode="gain-loss"/>
      </xsl:attribute>
  </xsl:attribute-set>

<xsl:template match="stock" mode="gain-loss">
  <xsl:choose>
     <xsl:when test="(current - paid) * qty >= 0">gain</xsl:when>
     <xsl:otherwise>loss</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="property" mode="gain-loss">
    <xsl:choose>
       <xsl:when test="appriasal - paid  >= 0">gain</xsl:when>
       <xsl:otherwise>loss</xsl:otherwise>
    </xsl:choose>
</xsl:template>

<xsl:template match="portfolio">
    <html>
     <head>
      <title>Мой портфель</title>
      <link  rel="stylesheet"  type="text/css"  href="portfolio.css"/>
     </head>

     ...

</xsl:template>

<xsl:template match="stock">
  <tr>
    <td><xsl:value-of select="symbol"/></td>
    <td align="right"><xsl:value-of select="current"/></td>
    <td align="right"><xsl:value-of select="paid"/></td>
    <td align="right"><xsl:value-of select="qty"/></td>
    <td align="right" xsl:use-attribute-sets="gain-loss">
      <xsl:value-of
```

```
            select="format-number((current - paid) * qty, '#,##0.00')"/>
      </td>
    </tr>
  </xsl:template>

  <xsl:template match="property">
    <tr>
      <td><xsl:value-of select="address"/></td>
      <td align="right"><xsl:value-of select="paid"/></td>
      <td align="right"><xsl:value-of select="appriasal"/></td>
      <td align="right" xsl:use-attribute-sets="gain-loss">
        <xsl:value-of
              select="format-number(appriasal - paid, '#,##0.00')"/>
      </td>
    </tr>
  </xsl:template>

  </xsl:stylesheet>
```

Позже главный по стилизации решит, как отображать данные в ячейках `<td class="gain">` и `<td class="loss">`, подготовив файл *portfolio.css*, как показано в примере 10.1.

Пример 10.1. portfolio.css

```
td.gain
{
    color:black;
}

td.loss
{
    color:red;
    font-weight:700;
}
```

10.6. Создание замкнутого преобразования

Задача

Требуется упаковать и XML-данные, и таблицу стилей для их преобразования в HTML в один файл.

Решение

Для этого рецепта необходим браузер, поддерживающий XSLT-преобразования на стороне клиента (IE 6.0 и старше, IE 5.x + MSXML 3.0, Netscape

Navigator 6.1 и старше, Mozilla, Firefox 1.0 и старше, Apple Macintosh Safari (Tiger), и т.д.):

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

<?xml-stylesheet type="application/xml" href="selfcontained.xsl"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:pf="http://www.ora.com/XSLTCookbook/namespaces/portfolio">

<portfolio xmlns="http://www.ora.com/XSLTCookbook/namespaces/portfolio">
  <investment>
    <symbol>IBM</symbol>
    <current>72.70</current>
    <paid>65.00</paid>
    <qty>1000</qty>
  </investment>
  <investment>
    <symbol>JMAR</symbol>
    <current>1.90</current>
    <paid>5.10</paid>
    <qty>5000</qty>
  </investment>
  <investment>
    <symbol>DELL</symbol>
    <current>24.50</current>
    <paid>18.00</paid>
    <qty>100000</qty>
  </investment>
  <investment>
    <symbol>P</symbol>
    <current>57.33</current>
    <paid>63</paid>
    <qty>100</qty>
  </investment>
</portfolio>

<xsl:output method="html" />

  <xsl:attribute-set name="gain-loss-font">
  <xsl:attribute name="color">
     <xsl:choose>
      <xsl:when test="(pf:current - pf:paid) * pf:qty >= 0">black</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
     </xsl:choose>
```

```
  </xsl:attribute>
  </xsl:attribute-set>

<xsl:template match="xsl:stylesheet">
  <xsl:apply-templates select="pf:portfolio"/>
</xsl:template>

<xsl:template match="pf:portfolio">
  <html>
    <head>
      <title>Мой портфель</title>
    </head>

    <body bgcolor="#FFFFFF" text="#000000">
      <h1>Портфель</h1>
      <table border="1" cellpadding="2">
        <tbody>
          <tr>
            <th>Символ</th>
            <th>Текущая цена</th>
            <th>Цена на момент покупки</th>
            <th>Количество</th>
            <th>Прибыли/Убытки</th>
          </tr>
        <xsl:apply-templates/>
      </tbody>
     </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="pf:investment">
  <tr>
  <td><xsl:value-of select="pf:symbol"/></td>
  <td><xsl:value-of select="pf:current"/></td>
  <td><xsl:value-of select="pf:paid"/></td>
  <td><xsl:value-of select="pf:qty"/></td>
  <td><font xsl:use-attribute-sets="gain-loss-font"><xsl:value-of select=
"format-number((pf:current - pf:paid) * pf:qty, '#,##0.00')"/></font></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

В работе этой таблицы стилей принимают участие два компонента.

Первый – команда обработки `xml-stylesheet`, которая говорит браузеру, что таблица стилей и данные, которые она обрабатывает, находятся в одном и том же документе. Сослаться на тот же документ, в котором находится таблица стилей, можно с помощью атрибута `href=""`, не указывая имени файла. Это полезно, если в дальнейшем вы захотите переименовать файл.

Второй – шаблон, который сопоставляется с элементом `xsl:stylesheet` и перенаправляет обработку на внедренные XML-данные. В данном случае элементы находятся в пространстве имен *http://www.ora.com/XSLTCookbook/namespaces/portfolio*.

Обсуждение

Этот рецепт представляет собой трюк, которым можно поражать друзей. В каком-то смысле перемешивание содержимого и стилизации в одном документе противоречит самому духу технологии. Однако иногда иметь единственный файл удобно, поэтому нечего стыдиться, если этот рецепт отвечает вашим целям.

Официально для достижения такого результата следует внедрять таблицу стилей в документ, а не наоборот. Подробности см. в разделе *http://www.w3.org/TR/xslt#section-Embedding-Stylesheets* спецификации XSLT 1.0 или в разделе *http://www.w3.org/ TR/xslt20/#embedded* для XSLT 2.0. К сожалению, IE пока не поддерживает внедренных таблиц стилей, поэтому приходится идти в обход.

Для доставки содержимого в такой форме не обязательно помещать его непосредственно в файл вместе с таблицей стилей. Ниже показано, как объединить таблицу стилей и XML-файл в замкнутый пакет. Требуется лишь, чтобы XML-данные находились в некотором пространстве имен, а таблица стилей начинала обработку не с корневого элемента (/).

```
<!-- generate-selfcontained.xslt -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xso="dummy">

<!-- Повторно используем тождественное преобразование -->
<xsl:import href="../util/copy.xslt"/>

<!-- Эта таблица стилей генерирует другую таблицу стилей,
поэтому используем префикс xso в качестве псевдонима xsl -->
<xsl:namespace-alias stylesheet-prefix="xso" result-prefix="xsl"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:strip-space elements="*"/>
<!-- Выбрасывать пробелы из текстовых узлов не стоит -->
```

```
<xsl:preserve-space elements="xsl:text"/>

<!-- Имя файла, содержащего xml-данные -->
<xsl:param name="datafile"/>
<!-- Имя результирующего файла -->
<xsl:param name="outfile"/>

<xsl:template match="/">
  <!-- Вставляем команду обработки, которая сообщает браузеру о том,
  что $outfile — таблица стилей -->
  <xsl:processing-instruction name="xml-stylesheet">
    <xsl:text>type="application/xml" href="</xsl:text>
    <xsl:value-of select="$outfile"/>"<xsl:text/>
  </xsl:processing-instruction>

  <xsl:apply-templates/>

</xsl:template>

<xsl:template match="xsl:stylesheet">

  <xsl:copy>
    <xsl:copy-of select="@*"/>

    <xsl:apply-templates/>

      <!-- Generate the xslt that tells the
    <xso:template match="xsl:stylesheet">
       <xso:apply-templates select="{name(document($datafile)/*)}"/>
    </xso:template>

    <!-- Вставляем данные -->
    <xsl:copy-of select="document($datafile)"/>

  </xsl:copy>

</xsl:template>

</xsl:stylesheet>
```

Эту таблицу стилей можно использовать для преобразования другой таблицы и ассоциированных с ней данных в замкнутый документ, осуществляющий преобразование в HTML. Исходным файлом должна быть таблица стилей, которой имя файла данных передается в виде параметра `$datafile`. Необходим еще параметр `$outfile`, чтобы правильно генерировалась команда обработки `xml-stylesheet`.

Для генерации выходного файла можно запустить Saxon из командной строки:

```
saxon -o self-contained.xsl pf-portfolio.xslt generate-selfcontained.xslt
    datafile="pf-portfolio.xml" outfile="self-contained.xsl"
```

Здесь *self-contained.xsl* – имя результирующей таблицы стилей, а *pf-portfolio.xslt* –таблица стилей, объединяемая с файлом данных *pf-portfolio.xml*.

10.7. Заполнение формы

Задача

Требуется объединить XML-данные с формой перед отправкой ее клиенту.

Решение

Если HTML-документ соответствует спецификации XHTML или хотя бы написан с учетом требований XML, то с помощью XSLT-преобразования его можно объединить с данными, хранящимися в формате XML. В этом рецепте мы объединим данные из XML-документа с шаблонной HTML-формой. Представьте, например, что дизайнер подготовил форму, которую пользователь должен заполнить перед совершением покупки в Интернет-магазине. В эту форму нужно добавить налог с продаж, величина которого зависит от штата, указанного пользователем при вводе адреса для выставления счета. Поскольку и штат, в пользу которого компания взимает налог, и ставка налога могут изменяться, «зашивать» их в форму было бы неправильно. Вместо этого сервер мог бы подставить нужные данные в форму динамически, применив XSLT-преобразование.

Данные о ставках налога можно хранить (или извлекать из базы данных) в таком виде:

```
<salesTax>
  <state>
      <name>AL</name>
      <tax>4</tax>
  </state>
  <state>
      <name>AK</name>
      <tax>0</tax>
  </state>
  <state>
      <name>AZ</name>
      <tax>5.6</tax>
  </state>

...

  <state>
```

```
      <name>WY</name>
      <tax>4</tax>
  </state>
</salesTax>
```

А заготовка формы может выглядеть так:

```
<html>
  <head>
   <title>Check Out</title>
   <script type="text/javascript" language="JavaScript">
   <!--
   /* Инициализировать налог для штата по умолчанию */
   setTax(document.customerInfo.billState)

   /* Пересчитать налог при изменении штата */
     function setTax(field)
       {
    var value = new String(field.value)
    var commaPos = value.indexOf(",")
    var taxObj = document.customerInfo.tax
    var tax = value.substr(commaPos + 1)
    var subtotalObj = document.customerInfo.subtotal
    taxObj.value = tax
    document.customerInfo.total.value =
          parseFloat(subtotalObj.value) +
          (parseFloat(subtotalObj.value) * parseFloat(tax) / 100.00)
     }
   -->
   </script>
  </head>

<body bgcolor="#FFFFFF" text="#000000">
<h1>Расчет</h1>
<form name="customerInfo" method="post" action="">
  <table width="70%" border="0" cellspacing="3" cellpadding="3">
    <tr>
      <td width="7%"> </td>
      <td width="32%">
        <div align="center"><b>Адрес доставки</b></div>
      </td>
      <td width="20%"> </td>
      <td width="7%"> </td>
      <td width="34%">
        <div align="center"><b>Адрес для выставления счета</b></div>
```

```
  </td>
</tr>
<tr>
  <td width="7%">Фамилия</td>
  <td width="32%">
   <input type="text" name="shipName" maxlength="40" size="50" border="0"
   align="absmiddle"/>
 </td>
 <td width="20%"> </td>
 <td width="7%">Фамилия</td>
 <td width="34%">
   <input type="text" name="billName" maxlength="40" size="50" border="0"
   align="absmiddle"/>
 </td>
</tr>
<tr>
  <td width="7%">Адрес</td>
  <td width="32%">
   <input type="text" name="shipAddr" maxlength="40" size="50" border="0"
    align="absmiddle"/>
  </td>
  <td width="20%"> </td>
  <td width="7%">Адрес</td>
  <td width="34%">
   <input type="text" name="billAddr" maxlength="40" size="50" border="0"
    align="absmiddle"/>
  </td>
</tr>
<tr>
  <td width="7%">Город</td>
  <td width="32%">
   <input type="text" name="shipCity" maxlength="40" size="50" border="0"
   align="absmiddle"/>
  </td>
  <td width="20%"> </td>
  <td width="7%">Город</td>
  <td width="34%">
   <input type="text" name="billCity" maxlength="40" size="50" border="0"
   align="absmiddle"/>
  </td>
</tr>
<tr>
  <td width="7%">Штат</td>
  <td width="32%">
```

```
    <select name="shipState" size="1" align="absmiddle">
    </select>
  </td>
  <td width="20%"> </td>
  <td width="7%">Штат</td>
  <td width="34%">
     <select name="billState" size="1" align="absmiddle"
                              onChange="setTax(this)">
    </select>
  </td>
</tr>
<tr>
  <td width="7%">Почтовый индекс</td>
  <td width="32%">
   <input type="text" name="shipZip" maxlength="10" size="15" border="0"
    align="absmiddle"/>
  </td>
  <td width="20%"> </td>
  <td width="7%">Почтовый индекс</td>
  <td width="34%">
   <input type="text" name="billZip" maxlength="10" size="15" border="0"
    align="absmiddle"/>
  </td>
</tr>
<tr>
  <td width="7%"> </td>
  <td width="32%"> </td>
  <td width="20%"> </td>
  <td width="7%"> </td>
  <td width="34%"> </td>
</tr>
<tr>
  <td width="7%"> </td>
  <td width="32%"> </td>
  <td width="20%"> </td>
  <td width="7%">Сумма</td>
  <td width="34%">
    <input type="text" name="subtotal" readonly="1" value="100.00"/>
  </td>
</tr>
<tr>
  <td width="7%"> </td>
  <td width="32%"> </td>
  <td width="20%"> </td>
```

```
        <td width="7%">Налог</td>
        <td width="34%">
          <input type="text" name="tax" readonly="1"/>
        </td>
      </tr>
      <tr>
        <td width="7%"> </td>
        <td width="32%"> </td>
        <td width="20%"> </td>
        <td width="7%">Итого</td>
        <td width="34%">
          <input type="text" name="total" readonly="1"/>
        </td>
      </tr>
    </table>
  </form>
  </body>
  </html>
```

Объединение сводится к преобразованию, в котором действие по умолчанию – это копирование HTML-содержимого в выходной файл (см. рецепт 6.5). Встретив же элементы `select` для штатов в адресе доставки и выставления счета, преобразование вставляет данные из XML-документа в виде элементов `option`. Чтобы не усложнять пример, я включил и название штата, и ставку налога в атрибут `value` элемента `option`, но можно было бы написать функцию сопоставления на JavaScript.

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

 <xsl:import href="../util/copy.xslt"/>

 <xsl:output method="html"/>

 <xsl:template match="html">
   <xsl:copy>
      <xsl:apply-templates/>
   </xsl:copy>
 </xsl:template>

 <xsl:template match="select[@name='shipState' or @name='billState']">
   <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:for-each select="document('salesTax.xml')/salesTax/state">
          <option value="{name}',',{tax}">
              <xsl:value-of select="name"/>
```

```
            </option>
          </xsl:for-each>
        </xsl:copy>
      </xsl:template>

    </xsl:stylesheet>
```

Обсуждение

Понятно, что для решения задачи, рассмотренной в этом примере, существуют и другие, быть может, более подходящие технологии (на ум сразу приходят ASP и JSP). Однако есть два аспекта, которые, возможно, заставят вас предпочесть решение на основе XSLT.

Во-первых, отметим, что заготовка формы – это стандартный HTML-код, в нем нет никаких серверных элементов. Небольшой фрагмент клиентского сценария имеется, но его можно вставить и в процессе XSLT-преобразования. Автору заготовки ничего не нужно знать о технологии, с помощью которой форма будет заполняться на стороне сервера. Вообще, у автора этой Web-страницы может не быть никакого опыта программирования. С другой стороны, автору преобразования нет нужды что-то знать об HTML или графическом дизайне[1]. Они лишь должны договориться между собой о содержимом и схеме именования. Таким образом, XSLT-преобразование динамического контента обеспечивает настоящее разделение обязанностей, чего так недостает другим методикам.

Во-вторых, с помощью преобразований можно не просто вставлять содержимое; его можно частично удалить или реорганизовать, не замусоривая HTML-код не свойственными ему конструкциями.

См. также

Ограничения по времени и месту не позволили мне рассмотреть в этой книге XForms, но интересующемуся читателю я настоятельно рекомендую познакомиться с этой новой технологией (*http://www.w3.org/MarkUp/Forms/*). В документах консорциума W3C XForms описывается следующим образом:

Используемые в настоящий момент Web-формы не обеспечивают отделения содержимого формы от его представления. Напротив, в XForms-форме предусмотрены разделы, описывающие, для чего форма предназначена и как она выглядит. Это позволяет гибко присоединять к XML-определению формы различные способы представления, в том числе и классические XHTML-формы.

Перечислим основные задачи, которые призвана решать технология XForms:

- поддержка браузеров в карманных, телевизионных и настольных устройствах, а также принтеров и сканеров;
- развитый пользовательский интерфейс, отвечающий потребностям бизнеса, бытового потребителя и приложений для управления устройствами;

[1] Это как раз относится ко мне самому.

- разделение данных, логики и представления;
- улучшенная поддержка интернационализации;
- поддержка структурирования данных в форме;
- развитая логика работы форм;
- поддержка размещения нескольких форм на странице и многостраничных форм;
- поддержка операций приостановки и возобновления;
- естественная интеграция с другими наборами XML-тегов.

Глава 11. Преобразование XML в SVG

Сюжет фотографии можно найти повсюду.
Надо лишь уметь замечать и выстраивать.
Эллиот Эрвитт

11.0. Введение

Scalable Vector Graphics (SVG) – это формат векторной графики, записываемый в синтаксисе XML. У него есть все возможности революционно изменить способ доставки графического контента через Интернет. Одна из самых убедительных причин кодировать графику в виде XML заключается в том, что рендеринг данных оказывается частным случаем преобразования. Поэтому язык XSLT, в который не включены никакие графические возможности, способен генерировать сложные образы, так как большая часть работы делегируется интерпретатору SVG, работающему внутри браузера.

Хотя в этой главе предполагается, что читатель знаком с SVG, мы все же кратко опишем некоторые простые факты, которыми будем пользоваться.

Первое, что нужно знать, приступая к работе с графической системой, – как в ней устроена система координат. Потратив годы на изучение алгебры, тригонометрии и математического анализа, многие технически подкованные читатели склонны считать наиболее естественной декартову систему координат. В ней координата x (абсцисса) увеличивается слева направо, а координата y (ордината) – снизу вверх. Увы, в SVG применяется не декартова система. Направление оси y изменено на противоположное, то есть точка (0,0) находится в левом верхнем углу, а ордината увеличивается сверху вниз. Для многих приложений неважно, как организована система координат. Но для отображения графических данных декартова система удобнее, поскольку ориентация осей на дисплее соответствует интуитивным ожиданиям пользователя. В SVG есть мощный механизм, позволяющий трансформировать систему координат в соответствии с потребностями приложения. Для этого к отдельным линиям или формам, а также группам графических элементов применяются геометрические преобразования: параллельный перенос, поворот и гомотетия (масштабирование). В частности, указать, что вы собираетесь работать в декартовой системе, можно, задав следующее преобразование:

```
<svg:g transform="translate(0,{$height}) scale(1,-1)">
    <!-- Все содержимое представлено в декартовых координатах -->
</svg:g>
```

Здесь `$height` – высота всего SVG-изображения или наибольшая ордината по всем входящим в него графическим объектам.

При вычерчивании графика можно перенести начало координат и масштабировать изображение в соответствии с данными:

```
<svg:g transform="scale(1,{$height div $max})">
    <!-- Все содержимое представлено в декартовых координатах -->
</svg:g>
```

В этом примере координата y масштабируется с учетом высоты SVG-изображения `$height` и максимального представимого на графике значения `$max`.

Преобразование в декартову систему координат и масштабирование удобны для представления данных на графике, но при попытке разместить в новой системе текст мы сталкиваемся с трудностями. Вследствие изменения направления осей текст рисуется вверх ногами, а из-за масштабирования перекашивается. Поэтому к тексту следует применить компенсирующее преобразование:

```
<svg:text    x="{$someXPos}"
             y="{$someYPos}"
             transform="translate({$someXPos},{$someYPos})
                        scale(1,{-$max div $height})
                        translate({-$someXPos},{-$someYPos})">
Текст
</svg:text>
```

От таких преобразований у меня голова идет кругом, но иногда это лучший способ добиться нужного результата.

11.1. Преобразование имеющейся заготовки SVG

Задача

Требуется графически отобразить данные, заполнив ими имеющийся SVG-шаблон.

Решение

XSLT 1.0

Пусть требуется вывести данные в виде столбчатой диаграммы. Можно представить себе XSLT-преобразование, которое конструирует SVG-представление такой диаграммы с нуля (см. рецепт 11.2). Однако человеку, которой не очень уверенно владеет графическими манипуляциями, эта задача может показаться пугающе сложной. Но иногда нужно лишь вывести данные в нескольких фиксированных точках. В таком случае проще заранее создать в SVG-редакторе изображение, которое будет служить шаблоном, в который мы подставим конкретные данные с помощью XSLT. При таком подходе задача существенно упрощается.

Рассмотрим следующий шаблон столбчатой диаграммы:

```
<svg width="650" height="500">
  <g id="axis" transform="translate(0 500) scale(1 -1)">
    <line id="axis-y" x1="30" y1="20" x2="30" y2="450"
          style="fill:none;stroke:rgb(0,0,0);stroke-width:2"/>
    <line id="axis-x" x1="30" y1="20" x2="460" y2="20"
          style="fill:none;stroke:rgb(0,0,0);stroke-width:2"/>
  </g>
  <g id="bars" transform="translate(30 479) scale(1 -430)">
    <rect x="30" y="0" width="50" height="0.25"
          style="fill:rgb(255,0,0);stroke:rgb(0,0,0);stroke-width:0"/>
    <rect x="100" y="0" width="50" height="0.5"
          style="fill:rgb(0,255,0);stroke:rgb(0,0,0);stroke-width:0"/>
    <rect x="170" y="0" width="50" height="0.75"
          style="fill:rgb(255,255,0);stroke:rgb(0,0,0);stroke-width:0"/>
    <rect x="240" y="0" width="50" height="0.9"
          style="fill:rgb(0,255,255);stroke:rgb(0,0,0);stroke-width:0"/>
    <rect x="310" y="0" width="50" height="1"
          style="fill:rgb(0,0,255);stroke:rgb(0,0,0);stroke-width:0"/>
  </g>
  <g id="scale" transform="translate(29 60)">
    <text id="scale1" x="0px" y="320px"
          style="text-anchor:end;fill:rgb(0,0,0);font-size:10;font-family:
          Arial">0.25</text>
    <text id="scale2" x="0px" y="215px"
          style="text-anchor:end;fill:rgb(0,0,0);font-size:10;font-family:
          Arial">0.50</text>
    <text id="scale3" x="0px" y="107.5px"
          style="text-anchor:end;fill:rgb(0,0,0);font-size:10;font-family:
          Arial">0.75</text>
    <text id="scale4" x="0px" y="0px" style="text-anchor:end;fill:
          rgb(0,0,0);font-size:10;font-family:Arial">1.00</text>
  </g>
  <g id="key">
    <rect id="key1" x="430" y="80" width="25" height="15"
          style="fill:rgb(255,0,0);stroke:rgb(0,0,0);stroke-width:1"/>
    <rect id="key2" x="430" y="100" width="25" height="15"
          style="fill:rgb(0,255,0);stroke:rgb(0,0,0);stroke-width:1"/>
    <rect id="key3" x="430" y="120" width="25" height="15"
          style="fill:rgb(255,255,0);stroke:rgb(0,0,0);stroke-width:1"/>
    <rect id="key5" x="430" y="140" width="25" height="15"
          style="fill:rgb(0,255,255);stroke:rgb(0,0,0);stroke-width:1"/>
    <rect id="key4" x="430" y="160" width="25" height="15"
          style="fill:rgb(0,0,255);stroke:rgb(0,0,0);stroke-width:1"/>
```

```
    <text id="key1-text" x="465px" y="92px"
          style="fill:rgb(0,0,0);font-size:18;font-family:Arial">key1</text>
    <text id="key2-text" x="465px" y="112px"
          style="fill:rgb(0,0,0);font-size:18;font-family:Arial">key2</text>
    <text id="key3-text" x="465px" y="132px"
          style="fill:rgb(0,0,0);font-size:18;font-family:Arial">key3</text>
    <text id="key4-text" x="465px" y="152px"
          style="fill:rgb(0,0,0);font-size:18;font-family:Arial">key4</text>
    <text id="key5-text" x="465px" y="172px"
          style="fill:rgb(0,0,0);font-size:18;font-family:Arial">key5</text>
    </g>
    <g id="title">
        <text x="325px" y="20px" style="text-
anchor:middle;fill:rgb(0,0,0);
        font-size:24;font-family:Arial">Title</text>
    </g>
</svg>
```

На устройстве вывода он выглядит, как показано на рис. 11.1.
При создании этого шаблона мы руководствовались следующими соображениями.

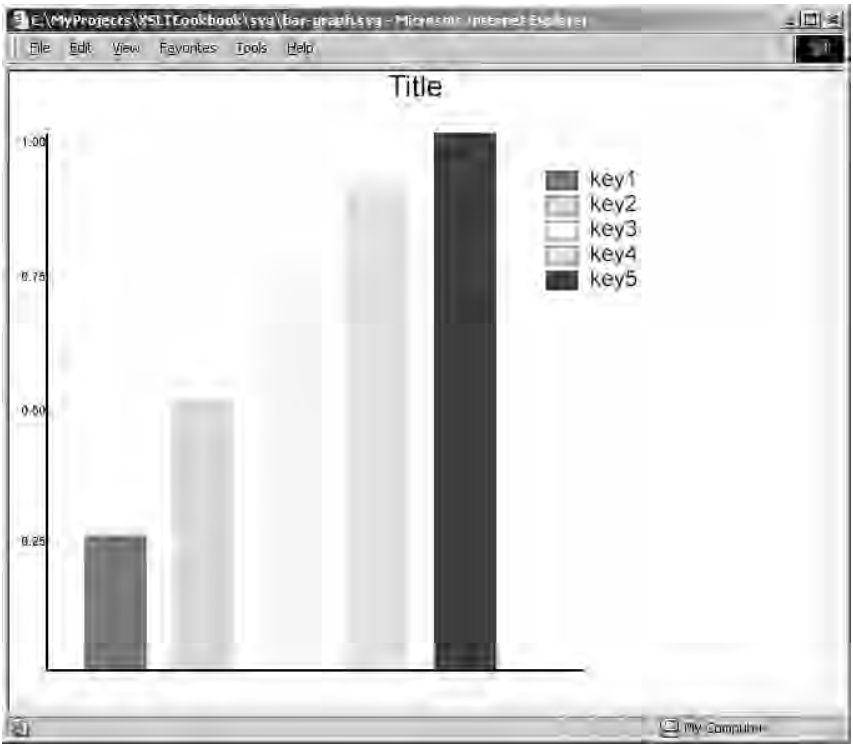


Рис. 11.1. SVG-шаблон столбчатой диаграммы

Во-первых, заранее известно, что предстоит вывести ровно пять значений, поэтому мы и создали пять столбцов. Столбцы помещены внутрь группы SVG (элемент g) с атрибутом `id="bars"`. Далее делается нечто важное – преобразование системы координат так, чтобы высота столбца представляла выводимое значение. Конкретно, преобразование `transform="translate(30 479) scale(1 -430)"`. `Translate` совмещает начало координат с началом столбцов. `Transform` меняет направление оси y и масштабирует ее так, что высота `height="1"` соответствует столбцу максимальной высоты. Отрицательное значение означает отражение (смену направления), а 430 определяет коэффициент масштабирования (430 – длина оси y). Конструкция `scale(1,-1)` внутри атрибута `transform` – это просто «заполнитель», который работает для фиктивных данных в SVG-графике. При обработке реальных данных таблица стилей подставит вместо него правильное значение.

Во-вторых, мы создали фиктивный элемент `g` с атрибутом `id="key"`. Требуется, чтобы порядок элементов внутри этой группы соответствовал порядку столбцов. Только в этом случае после преобразования график будет нарисован правильно.

В третьих, в группе `id="scale"` мы создали четыре текстовых элемента с подходящими промежутками для вывода делений шкалы по оси y. Каждому текстовому узлу сопоставлен атрибут `id`, а число, которым заканчивается его значение, равно количеству четвертей. Например, для текстового элемента 0,50 `id="scale2"`, потому что это деление представляет 2/4 (1/2) всей шкалы. В таблице стилей мы используем это число для отображения шкалы на значения реальных данных.

В-четвертых, мы создали фиктивный текстовый элемент внутри группы `title`. Представляемый им текст позиционируется в центре рисунка. И останется там после замены на реальный заголовок.

Тот факт, что все основные компоненты SVG-диаграммы находятся внутри групп, упрощает таблицу стилей, которая будет загружать данные для графика.

Мы собираемся представить в виде столбчатой диаграммы данные о продажах пяти самых продаваемых продуктов компании:

```
<product-sales description="Top 5 Product Sales (in $1000)">
  <product name="Widget">
    <sales multiple="1000" currency="USD">70</sales>
   </product>
  <product name="Foo Bar">
    <sales multiple="1000" currency="USD">880</sales>
  </product>
  <product name="Grunt Master 9000">
    <sales multiple="1000" currency="USD">1000</sales>
  </product>
  <product name="Spam Slicer">
    <sales multiple="1000" currency="USD">532</sales>
  </product>
  <product name="Wonder Bar">
    <sales multiple="1000" currency="USD">100</sales>
```

```
  </product>
</product-sales>
```

Показанная ниже таблица стилей объединяет SVG-шаблон с файлом данных, в результате чего создается график, на котором показаны реальные данные:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://www.exslt.org/math"
  exclude-result-prefixes="math">

<!-- По умолчанию SVG-шаблон копируется в выходной документ -->
<xsl:import href="../util/copy.xslt"/>

<!-- Необходимо найти максимальное значение.-->
<!-- Для масштабирования применяем шаблон max. -->
<xsl:include href="../math/math.max.xslt"/>

<!-- Имя файла данных передается в виде параметра. -->
<xsl:param name="data-file"/>

<!-- Выходной файл будет иметь тип SVG и соответствующую ему DTD-схему. -->
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
  doctype-public="-//W3C//DTD SVG 1.0/EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<!-- Загружаем данные в переменную-набор узлов, чтобы к ним было -->
<!-- проще обращаться -->
<xsl:variable name="bar-values" select="document($data-file)/*/*/sales"/>

<!-- Загружаем имена столбцов в переменную-набор узлов, чтобы к ним было -->
<!-- проще обращаться -->
<xsl:variable name="bar-names" select="document($data-file)/*/*/@name"/>

<!-- Находим максимальное значение -->
<xsl:variable name="max-data">
  <xsl:call-template name="math:max">
    <xsl:with-param name="nodes" select="$bar-values"/>
  </xsl:call-template>
</xsl:variable>

<!-- Из эстетических соображений масштабируем график так, чтобы -->
<!-- максимально представимое значение было на 10% больше реального -->
<!-- максимума -->
<xsl:variable name="max-bar" select="$max-data + $max-data div 10"/>

<!-- Поскольку каждому компоненту графика соответствует именованная
```

```
группа, таблицу стилей легко структурировать так, чтобы производилось
сопоставление с каждой группой и выполнялось подходящее преобразование. -->

<!-- Копируем группу делений и заменяем текстовые значения
соответствующими диапазону имеющихся данных. Для вычисления нужного
кратного 0.25 пользуемся числовой частью значения атрибута. -->

<xsl:template match="g[@id='scale']">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:for-each select="text">
      <xsl:copy>
      <xsl:copy-of select="@*"/>
        <xsl:variable name="factor"
        select="substring-after(@id,'scale') * 0.25"/>
        <xsl:value-of select="$factor * $max-bar"/>
       </xsl:copy>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>

<!-- В каждом компоненте key подменяем текстовые значения. -->
<xsl:template match="g[@id='key']">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:apply-templates select="rect"/>
    <xsl:for-each select="text">
    <xsl:variable name="pos" select="position(  )"/>
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:value-of select="$bar-names[$pos]"/>
    </xsl:copy>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>

<!-- Вместо заголовка подставляем извлеченное из данных описание.
Можно было бы вместо этого передавать заголовок в качестве параметра. -->
<xsl:template match="g[@id='title']">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:for-each select="text">
      <xsl:copy>
```

```
          <xsl:copy-of select="@*"/>
          <xsl:value-of select="document($data-file)/*/@description"/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>

  <!-- Для создания столбцов: -->
  <!-- 1) в атрибут transform подставляем преобразование масштабирования, -->
  <!--    вычисляемое по значению $max-bar -->
  <!-- 2) Подменяем значение высоты столбца -->
  <xsl:template match="g[@id='bars']">
  <xsl:copy>
    <xsl:copy-of select="@id"/>
    <xsl:attribute name="transform">
    <xsl:value-of select="concat('translate(60 479) scale(1 ',
      -430 div $max-bar,')')"/>
    </xsl:attribute>
    <xsl:for-each select="rect">
      <xsl:variable name="pos" select="position()"/>
      <xsl:copy>
        <xsl:copy-of select="@*"/>
        <xsl:attribute name="height">
          <xsl:value-of select="$bar-values[$pos]"/>
        </xsl:attribute>
      </xsl:copy>
    </xsl:for-each>
   </xsl:copy>
  </xsl:template>

  </xsl:stylesheet>
```

После применения этой таблицы стилей к SVG-шаблону получится диаграмма, показанная на рис. 1.2.

XSLT 2.0

Я не буду повторять все решение с модификациями для версии 2.0, а обращу внимание только на необходимые и желательные изменения.

Вместо шаблона max, написанного в главе 3, можно воспользоваться встроенной в XPath 2.0 функцией max.

```
<xsl:variable name="bar-values" select="document($data-file)/*/*/sales"
        as="xs:double*"/>
<!-- Находим максимальное значение -->
<xsl:variable name="max-data" select="max($bar-values)"/>
```

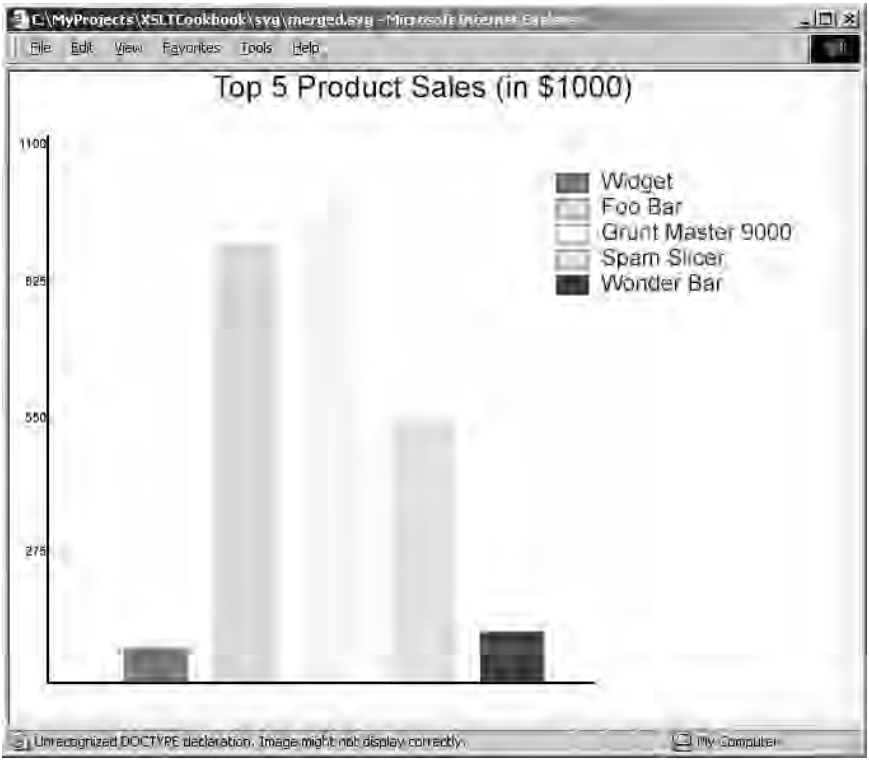


Рис. 11.2. Столбчатая диаграмма, сгенерированная с помощью XSLT

В XSLT 2.0 к типам надо относиться внимательно. Для преобразования строки в число и обратно необходимо пользоваться специальными функциями:

```
<xsl:template match="g[@id='scale']">
  <xsl:copy>
    <xsl:copy-of select="@*"/>
    <xsl:for-each select="text">
      <xsl:copy>
      <xsl:copy-of select="@*"/>
         <xsl:variable name="factor"
           select="number(substring-after(@id,'scale')) * 0.25"/>
           <xsl:value-of select="$factor * $max-bar"/>
        </xsl:copy>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Можно использовать более краткую форму записи `xsl:attribute`. Обратите также внимание на обязательное явное преобразование числа в строку:

```
<xsl:attribute name="transform" select="concat('translate(60 479) scale(1 ',
  string(-430 div $max-bar),')')"/>
```

Обсуждение

Создание SVG-документа непосредственно из данных требует большого объема математических вычислений. Это чревато ошибками и разочарованиями, особенно если работа с графикой – не ваш конек. В этом рецепте мы предложили обходной путь – создавать заготовку SVG в визуальном редакторе, а не программно. Кроме того, чтобы упростить преобразование данных в графическую форму, мы задействовали встроенный в SVG механизм трансформаций.

Конечно, у этого рецепта есть очевидные ограничения.

Во-первых, нужно заранее знать количество точек на графике. Более общий построитель столбчатых диаграмм мог бы вычислить количество столбцов и распределить их автоматически, анализируя данные на этапе выполнения. Эту проблему можно частично разрешить, создав SVG-шаблон, содержащий, скажем, десять столбцов, и удалить лишние во время выполнения. Но при таком подходе представление будет некрасивым; ведь если подставить в шаблон, рассчитанный на десять элементов, всего два, то останутся «дырки».

Во-вторых, хотя и можно создавать графики, более сложные, чем в этом простеньком примере, мы ограничены линейным отображением величин на их визуальные образы. Так, не вполне ясно, как можно было бы применить ту же технику к построению секторных диаграмм; для изменения площади сектора недостаточно подставить значение единственного атрибута.

Третье ограничение проистекает из того факта, что источник данных, SVG-шаблон и таблица стилей оказываются тесно связанными между собой. Поэтому всякий раз, как вы захотите воспользоваться этой техникой, придется создавать новый шаблон и новую таблицу стилей. В конечном итоге вы потратите на это больше времени, чем на поиск более общего решения с самого начала.

Несмотря на все ограничения, этот пример все же представляет ценность в качестве отправной точки для тех, кто только приступает к изучению SVG-графики. Точнее, он позволяет задействовать возможности SVG-редактора для предварительной подготовки изображений: выравнивания, равномерного распределения и пропорционального масштабирования.

11.2. Создание повторно используемых утилит генерации SVG для графиков и диаграмм

Задача

Требуется создать библиотеку генераторов SVG, которой можно было бы пользоваться в приложениях, связанных с графическим представлением данным.

Решение

XSLT 1.0

Если вы планируете много заниматься созданием SVG-файлов с помощью XSLT, то было бы полезно разработать библиотеку шаблонов для генерации графических компонентов. В этом разделе мы приведем несколько примеров такого рода.

Генерация осей

В этом примере создается набор общих шаблонов для генерации размеченных осей x и y.

```
<!-- Нарисовать ось X с делениями -->
<xsl:template  name="svgu:xAxis">
   <xsl:param  name="min"
               select="0"/>   <!-- Минимальная абсцисса -->
   <xsl:param  name="max"
               select="100"/> <!-- Макимальная абсцисса -->
   <xsl:param  name="offsetX"
               select="0"/>  <!--  Смещение от левой границы области -->
   <xsl:param  name="offsetY"
               select="0"/> <!-- Смещение от нижней границы области -->
   <xsl:param  name="width"
               select="500"/> <!--  Ширина физической области рисования -->
   <xsl:param  name="height"
               select="500"/> <!-- Высота физической области рисования -->
    <xsl:param name="majorTicks"
               select="10"/> <!-- Количество крупных делений по оси -->
   <xsl:param  name="majorBottomExtent"
               select="4"/> <!-- Длина части крупного деления под осью -->
   <xsl:param  name="majorTopExtent"
               select="$majorBottomExtent"/> <!-- Длина части крупного
                                                  деления над осью -->
   <xsl:param  name="labelMajor"
               select="true(  )"/> <!-- Если true, подписывать крупные
                                     деления -->
   <xsl:param  name="minorTicks"
               select="4"/> <!-- Количество мелких делений внутри
                                крупного -->
   <xsl:param  name="minorBottomExtent"
               select="2"/> <!-- Длина части мелкого деления под осью -->
   <xsl:param  name="minorTopExtent"
               select="$minorBottomExtent"/> <!-- Длина части мелкого
                                                  деления над осью  -->
   <xsl:param  name="context"/>  <!-- Определенный пользователем
```

```
                                        индикатор контекста для вызовов
                                        шаблона форматирования -->

  <!-- Вычислить диапазон и коэффициенты масштабирования -->
  <xsl:variable name="range" select="$max - $min"/>
  <xsl:variable name="scale" select="$width div $range"/>

 <!-- Определить декартову систему, задав смещение начала координат и -->
 <!-- масштабирование -->
 <svg:g transform="translate({$offsetX},{$offsetY+$height})
             scale({$scale},-1) translate({$min},0)">
 <!-- Провести прямую, представляющую ось  -->
 <svg:line x1="{$min}" y1="0" x2="{$max}"  y2="0">
  <xsl:attribute name="style">
   <!-- Вызвать шаблон, который можно переопределить для -->
   <!-- задания другого стиля рисования оси -->
    <xsl:call-template name="xAxisStyle">
       <xsl:with-param name="context" select="$context"/>
    </xsl:call-template>
   </xsl:attribute>
 </svg:line>

 <!-- Нарисовать деления и подписи -->
   <xsl:call-template name="svgu:ticks">
      <xsl:with-param name="xMajor1" select="$min"/>
      <xsl:with-param name="yMajor1" select="$majorTopExtent"/>
      <xsl:with-param name="xMajor2" select="$min"/>
      <xsl:with-param name="yMajor2" select="-$majorBottomExtent"/>
      <xsl:with-param name="labelMajor" select="$labelMajor"/>
      <xsl:with-param name="freq" select="$minorTicks"/>
      <xsl:with-param name="xMinor1" select="$min"/>
      <xsl:with-param name="yMinor1" select="$minorTopExtent"/>
      <xsl:with-param name="xMinor2" select="$min"/>
      <xsl:with-param name="yMinor2" select="-$minorBottomExtent"/>
      <xsl:with-param name="nTicks"
                      select="$majorTicks * $minorTicks + 1"/>
      <xsl:with-param name="xIncr"
          select="($max - $min) div ($majorTicks * $minorTicks)"/>
      <xsl:with-param name="scale" select="1 div $scale"/>
    </xsl:call-template>
 </svg:g>

  </xsl:template>

<xsl:template name="svgu:yAxis">
```

```
<xsl:param name="min"
           select="0"/>   <!-- Минимальная ордината -->
<xsl:param name="max"
           select="100"/> <!-- Максимальная ордината -->
<xsl:param name="offsetX"
           select="0"/>   <!-- Смещение от левой границы области -->
<xsl:param name="offsetY"
           select="0"/>   <!-- Смещение от правой границы области -->
<xsl:param name="width"
           select="500"/> <!-- Ширина физической области рисования -->
<xsl:param name="height"
           select="500"/> <!-- Высота физической области рисования -->
<xsl:param name="majorTicks"
           select="10"/>   <!-- Количество крупных делений по оси -->
<xsl:param name="majorLeftExtent"
           select="4"/> <!-- Длина части крупного деления слева
                             от оси -->
<xsl:param name="majorRightExtent"
           select="$majorBottomExtent"/> <!-- Длина части крупного
                                              деления справа от оси -->
<xsl:param name="labelMajor"
           select="true(  )"/> <!-- Если true, подписывать крупные
                                    деления -->
<xsl:param name="minorTicks"
           select="4"/>      <!-- Количество мелких делений внутри
                                  крупного -->
<xsl:param name="minorLeftExtent"
           select="2"/>      <!-- Длина части мелкого деления слева
            от оси -->
<xsl:param name="minorRightExtent"
           select="$minorBottomExtent"/> <!-- Длина части мелкого
                                              деления справа от оси -->
<xsl:param name="context"/>   <!-- Определенный пользователем
                                   индикатор контекста для вызовов
                                   шаблона форматирования -->

<xsl:param name="majorLeftExtent"
           select="4"/>
<xsl:param name="majorRightExtent"
           select="$majorLeftExtent"/>
<xsl:param name="minorLeftExtent"
           select="2"/>
<xsl:param name="minorRightExtent"
           select="$minorLeftExtent"/>
```

```
  <!-- Вычислить диапазон и коэффициенты масштабирования -->
  <xsl:variable name="range" select="$max - $min"/>
  <xsl:variable name="scale" select="$height div $range"/>

  <!-- Определить декартову систему, задав смещение начала координат и -->
  <!-- масштабирование -->
  <svg:g transform="translate({$offsetX},{$offsetY+$height})
                scale(1,{-$scale}) translate(0,{-$min})">
    <svg:line x1="0" y1="{$min}" x2="0"  y2="{$max}">
      <xsl:attribute name="style">
        <xsl:call-template name="yAxisStyle">
          <xsl:with-param name="context" select="$context"/>
        </xsl:call-template>
      </xsl:attribute>
    </svg:line>

<xsl:call-template name="svgu:ticks">
   <xsl:with-param name="xMajor1" select="-$majorLeftExtent"/>
   <xsl:with-param name="yMajor1" select="$min"/>
   <xsl:with-param name="xMajor2" select="$majorRightExtent"/>
   <xsl:with-param name="yMajor2" select="$min"/>
   <xsl:with-param name="labelMajor" select="$labelMajor"/>
   <xsl:with-param name="freq" select="$minorTicks"/>
   <xsl:with-param name="xMinor1" select="-$minorLeftExtent"/>
   <xsl:with-param name="yMinor1" select="$min"/>
   <xsl:with-param name="xMinor2" select="$minorRightExtent"/>
   <xsl:with-param name="yMinor2" select="$min"/>
   <xsl:with-param name="nTicks"
                   select="$majorTicks * $minorTicks + 1"/>
   <xsl:with-param name="yIncr"
         select="($max - $min) div ($majorTicks * $minorTicks)"/>
   <xsl:with-param name="scale" select="1 div $scale"/>
    </xsl:call-template>
 </svg:g>

</xsl:template>

<!-- Рекурсивный шаблон для рисования делений и подписей -->
<xsl:template name="svgu:ticks">
  <xsl:param name="xMajor1" />
  <xsl:param name="yMajor1" />
  <xsl:param name="xMajor2" />
  <xsl:param name="yMajor2" />
  <xsl:param name="labelMajor"/>
```

```
<xsl:param name="freq" />
<xsl:param name="xMinor1" />
<xsl:param name="yMinor1" />
<xsl:param name="xMinor2" />
<xsl:param name="yMinor2" />
<xsl:param name="nTicks" select="0"/>
<xsl:param name="xIncr" select="0"/>
<xsl:param name="yIncr" select="0"/>
<xsl:param name="i" select="0"/>
<xsl:param name="scale"/>
<xsl:param name="context"/>

<xsl:if test="$i &lt; $nTicks">
  <xsl:choose>
    <!-- Рисуем крупное деление -->
    <xsl:when test="$i mod $freq = 0">
      <svg:line x1="{$xMajor1}" y1="{$yMajor1}"
            x2="{$xMajor2}" y2="{$yMajor2}">
      </svg:line>
      <xsl:if test="$labelMajor">
        <xsl:choose>
```

Далее рисуются деления по осям x и y. В этом примере форматная строка «зашита» в код, чтобы не передавать еще один параметр, но можно было бы передать ее параметром в этот или в отдельный шаблон для форматирования:

```
<!-- Деления по оси x -->
<xsl:when test="$xIncr > 0">
   <!-- При выводе подписи нужно компенсировать искажение системы
   координат -->
     <svg:text x="{$xMajor1}" y="{$yMajor2}"
           transform="translate({$xMajor1},{$yMajor2})
                   scale({$scale},-1)
                   translate({-$xMajor1},{-$yMajor2})">
        <xsl:attribute name="style">
        <xsl:call-template name="xAxisLabelStyle">
           <xsl:with-param name="context"
                           select="$context"/>
        </xsl:call-template>
       </xsl:attribute>
     <!-- Может быть, формат подписи следовало сделать параметром -->
     <xsl:value-of select="format-number($xMajor1,'#0.0')"/>
  </svg:text>
 </xsl:when>
<!-- Деления по оси y -->
```

```
      <xsl:otherwise>
           <svg:text x="{$xMajor1}" y="{$yMajor1}"
                     transform="translate({$xMajor1},{$yMajor1})
                                scale(1,{-$scale})
                                 translate({-$xMajor1},{-$yMajor1})">
             <xsl:attribute name="style">
               <xsl:call-template name="yAxisLabelStyle">
                 <xsl:with-param name="context" select="$context"/>
               </xsl:call-template>
             </xsl:attribute>
               <xsl:value-of select="format-number($yMajor1,'#0.0')"/>
           </svg:text>
         </xsl:otherwise>
       </xsl:choose>
     </xsl:if>
   </xsl:when>
      <!-- Рисуем мелкие деления -->
   <xsl:otherwise>
        <svg:line x1="{$xMinor1}" y1="{$yMinor1}"
              x2="{$xMinor2}" y2="{$yMinor2}">
       </svg:line>
   </xsl:otherwise>
 </xsl:choose>

       <!-- Рекурсивный вызов для рисования следующего деления  -->
      <xsl:call-template name="svgu:ticks">
        <xsl:with-param name="xMajor1" select="$xMajor1 + $xIncr"/>
        <xsl:with-param name="yMajor1" select="$yMajor1 + $yIncr"/>
        <xsl:with-param name="xMajor2" select="$xMajor2 + $xIncr"/>
        <xsl:with-param name="yMajor2" select="$yMajor2 + $yIncr"/>
        <xsl:with-param name="labelMajor" select="$labelMajor"/>
        <xsl:with-param name="freq" select="$freq"/>
        <xsl:with-param name="xMinor1" select="$xMinor1 + $xIncr"/>
        <xsl:with-param name="yMinor1" select="$yMinor1 + $yIncr"/>
        <xsl:with-param name="xMinor2" select="$xMinor2 + $xIncr"/>
        <xsl:with-param name="yMinor2" select="$yMinor2 + $yIncr"/>
        <xsl:with-param name="nTicks" select="$nTicks"/>
        <xsl:with-param name="xIncr" select="$xIncr"/>
        <xsl:with-param name="yIncr" select="$yIncr"/>
        <xsl:with-param name="i" select="$i + 1"/>
        <xsl:with-param name="scale" select="$scale"/>
        <xsl:with-param name="context" select="$context"/>
      </xsl:call-template>
    </xsl:if>
```

```
  </xsl:template>

  <!-- Переопределить этот шаблон, если нужно изменить стиль оси x -->
  <xsl:template name="xAxisStyle">
    <xsl:param name="context"/>
     <xsl:text>stroke-width:0.5;stroke:black</xsl:text>
  </xsl:template>

  <!-- Переопределить этот шаблон, если нужно изменить стиль оси y -->
  <xsl:template name="yAxisStyle">
    <xsl:param name="context"/>
     <xsl:text>stroke-width:0.5;stroke:black</xsl:text>
  </xsl:template>

  <!-- Переопределить этот шаблон, если нужно изменить стиль подписей
 под осью x -->
  <xsl:template name="xAxisLabelStyle">
    <xsl:param name="context"/>
    <xsl:text>text-anchor:middle; font-size:8;
              baseline-shift:-110%</xsl:text>
  </xsl:template>

  <!-- Переопределить этот шаблон, если нужно изменить стиль подписей
 слева от оси y -->
  <xsl:template name="yAxisLabelStyle">
    <xsl:param name="context"/>
    <xsl:text>text-anchor:end;font-size:8;baseline-shift:-50%</xsl:text>
  </xsl:template>
```

Следующая таблица стилей порождает оси x и y с крупными и мелкими делениями и соответствующими им подписями:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
  exclude-result-prefixes="svgu test">

  <xsl:import href="svg-utils.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
    doctype-public="-//W3C//DTD SVG 1.0/EN"
    doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/
svg10.dtd"/>
```

```
    <xsl:variable name="width" select="300"/>
    <xsl:variable name="height" select="300"/>
    <xsl:variable name="pwidth" select="$width * 0.8"/>
    <xsl:variable name="pheight" select="$height * 0.8"/>
    <xsl:variable name="offsetX" select="($width - $pwidth) div 2"/>
    <xsl:variable name="offsetY" select="($height - $pheight) div 2"/>

    <xsl:template match="/">
      <svg:svg width="{$width}" height="{$height}">
        <xsl:call-template name="svgu:xAxis">
           <xsl:with-param name="min" select="0"/>
           <xsl:with-param name="max" select="10"/>
           <xsl:with-param name="offsetX" select="$offsetX"/>
           <xsl:with-param name="offsetY" select="$offsetY"/>
           <xsl:with-param name="width" select="$pwidth"/>
           <xsl:with-param name="height" select="$pheight"/>
        </xsl:call-template>

        <xsl:call-template name="svgu:yAxis">
           <xsl:with-param name="min" select="0"/>
           <xsl:with-param name="max" select="10"/>
           <xsl:with-param name="offsetX" select="$offsetX"/>
           <xsl:with-param name="offsetY" select="$offsetY"/>
           <xsl:with-param name="width" select="$pwidth"/>
           <xsl:with-param name="height" select="$pheight"/>
        </xsl:call-template>

      </svg:svg>

    </xsl:template>

</xsl:stylesheet>
```

В результате получаем оси, изображенные на рис. 11.3.

Если задать длины частей крупных делений выступающих над и справа от оси равными соответственно полной высоте и ширине области рисования, то получится сетка, изображенная на рис. 11.4.

```
    <xsl:call-template name="svgu:xAxis">
       <xsl:with-param name="min" select="0"/>
       <xsl:with-param name="max" select="10"/>
       <xsl:with-param name="offsetX" select="$offsetX"/>
       <xsl:with-param name="offsetY" select="$offsetY"/>
       <xsl:with-param name="width" select="$pwidth"/>
       <xsl:with-param name="height" select="$pheight"/>
       <xsl:with-param name="majorTopExtent" select="$pheight"/>
       </xsl:call-template>
```

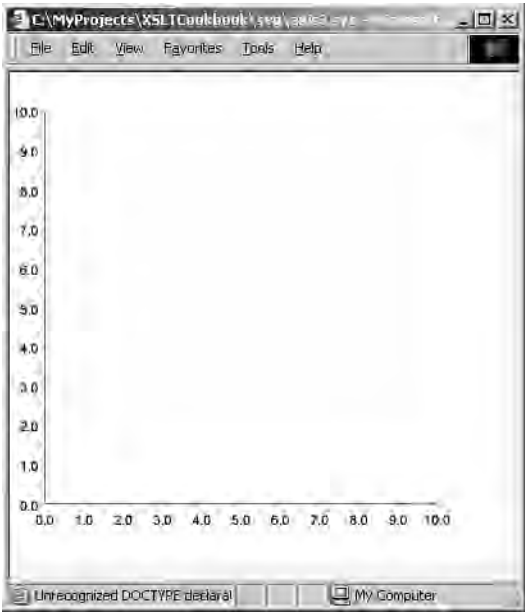


Рис. 11.3. Результат применения повторно используемой таблицы стилей для рисования осей

```
<xsl:call-template name="svgu:yAxis">
<xsl:with-param name="min" select="0"/>
<xsl:with-param name="max" select="10"/>
<xsl:with-param name="offsetX" select="$offsetX"/>
<xsl:with-param name="offsetY" select="$offsetY"/>
<xsl:with-param name="width" select="$pwidth"/>
<xsl:with-param name="height" select="$pheight"/>
<xsl:with-param name="majorRightExtent" select="$pwidth"/>
</xsl:call-template>
```

Если также поступить и с длинами мелких делений, то получится более мелкая сетка, изображенная на рис. 11.5.

```
<xsl:call-template name="svgu:xAxis">
        <xsl:with-param name="min" select="0"/>
        <xsl:with-param name="max" select="10"/>
        <xsl:with-param name="offsetX" select="$offsetX"/>
        <xsl:with-param name="offsetY" select="$offsetY"/>
        <xsl:with-param name="width" select="$pwidth"/>
        <xsl:with-param name="height" select="$pheight"/>
        <xsl:with-param name="majorTopExtent" select="$pheight"/>
        <xsl:with-param name="minorTopExtent" select="$pheight"/>
</xsl:call-template>
```

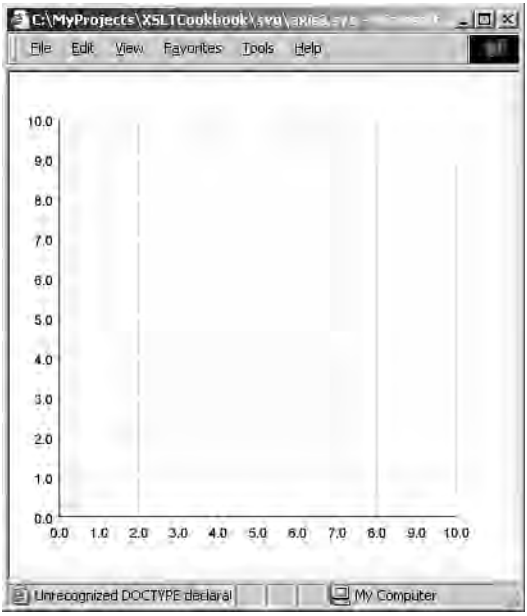


Рис. 11.4. Координатная сетка

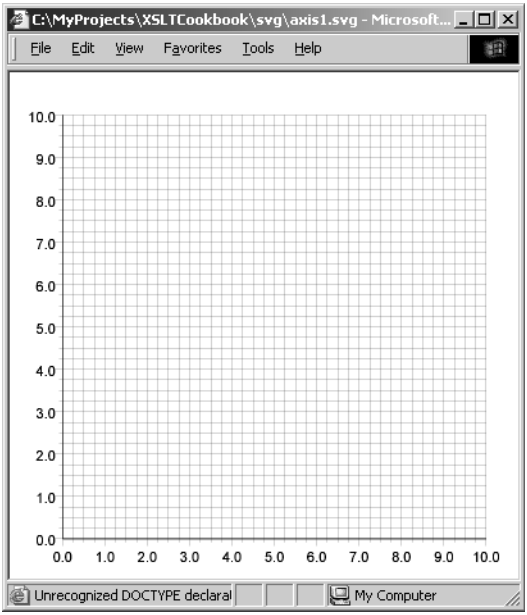


Рис. 11.5. Мелкая координатная сетка

```
<xsl:call-template name="svgu:yAxis">
       <xsl:with-param name="min" select="0"/>
       <xsl:with-param name="max" select="10"/>
```

```
            <xsl:with-param name="offsetX" select="$offsetX"/>
            <xsl:with-param name="offsetY" select="$offsetY"/>
            <xsl:with-param name="width" select="$pwidth"/>
            <xsl:with-param name="height" select="$pheight"/>
            <xsl:with-param name="majorRightExtent" select="$pwidth"/>
            <xsl:with-param name="minorRightExtent" select="$pwidth"/>
    </xsl:call-template>
```

Если сдвинуть оси и задать длины частей делений по обе стороны осей, то получатся четыре квадранта, как показано на рис. 11.6:

```
<xsl:call-template name="svgu:xAxis">
   <xsl:with-param name="min" select="-5"/>
   <xsl:with-param name="max" select="5"/>
   <xsl:with-param name="offsetX" select="0"/>
   <xsl:with-param name="offsetY" select="-$pheight div 2"/>
   <xsl:with-param name="width" select="$pwidth"/>
   <xsl:with-param name="height" select="$pheight"/>
   <xsl:with-param name="majorTopExtent" select="$pwidth div 2"/>
   <xsl:with-param name="majorBottomExtent" select="$pwidth div 2"/>
</xsl:call-template>

<xsl:call-template name="svgu:yAxis">
   <xsl:with-param name="min" select="-5"/>
   <xsl:with-param name="max" select="5"/>
```

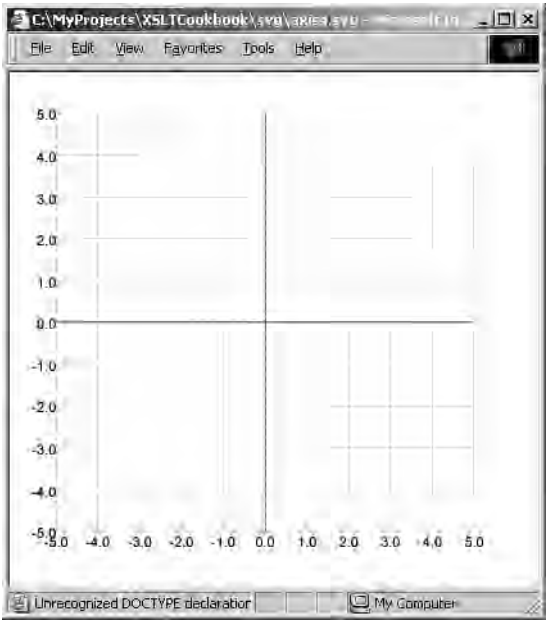


Рис. 11.6. Четыре квадранта и координатная сетка

```
      <xsl:with-param name="offsetX" select="-$pwidth div 2"/>
      <xsl:with-param name="offsetY" select="0"/>
      <xsl:with-param name="width" select="$pwidth"/>
      <xsl:with-param name="height" select="$pheight"/>
      <xsl:with-param name="majorRightExtent" select="$pwidth div 2"/>
      <xsl:with-param name="majorLeftExtent" select="$pwidth div 2"/>
  </xsl:call-template>
```

По умолчанию подписи располагаются снизу и слева от области рисования. Но можно придвинуть их к осям, переопределив два шаблона. Тогда получится результат, показанный на рис. 11.7.

```
<xsl:template name="xAxisLabelYOffset">
    <xsl:value-of select="-$pheight div 2"/>
</xsl:template>

<xsl:template name="yAxisLabelXOffset">
    <xsl:value-of select="$pwidth div 2"/>
</xsl:template>
```

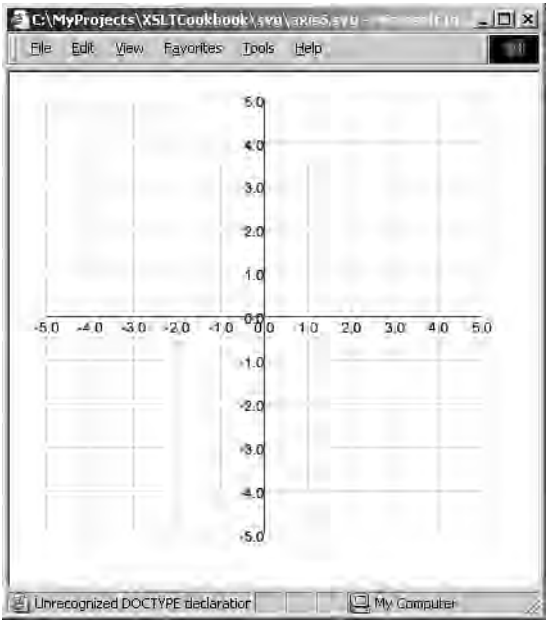


Рис. 11.7. Четыре квадранта с координатной сеткой и подписями на осях

Генерация столбцов

Данные часто представляются в виде столбчатых диаграмм, на которых можно провести наглядное сравнение. Напишем утилиту, которая будет порождать один столбец по переданному значению. Цвет и ширину столбца легко настроить.

В примере ниже столбцы можно также ориентировать в разных направлениях за счет поворота системы координат. Но при этом не следует забывать о компенсации поворота при выводе текста и о видимом порядке следования значений:

```
<xsl:template name="svgu:bars">
  <xsl:param name="data" select="/.."/>  <!-- данные для диаграммы -->
  <xsl:param name="width" select="500"/>
  <xsl:param name="height" select="500"/>
  <xsl:param name="orientation" select="0"/>
  <xsl:param name="barWidth" select="5"/>
  <xsl:param name="offsetX" select="0"/>
  <xsl:param name="offsetY" select="0"/>
  <xsl:param name="boundingBox" select="false(  )"/>
  <xsl:param name="barLabel" select="false(  )"/>
  <xsl:param name="max">
   <xsl:call-template name="emath:max">
     <xsl:with-param name="nodes" select="$data"/>
   </xsl:call-template>
  </xsl:param>
  <xsl:param name="context"/>

 <xsl:variable name="numBars" select="count($data)"/>
  <xsl:variable name="spacing" select="$width div ($numBars + 1)"/>

 <xsl:if test="$boundingBox">
  <svg:g transform="translate({$offsetX},{$offsetY})
                translate({$width div 2},{$height div 2})
                rotate({$orientation - 180})
                translate({-$width div 2},{-$height div 2})">
     <svg:rect x="0" y="0"
     height="{$height}" width="{$width}"
     style="stroke: black;
            stroke-width:0.5;stroke-opacity:0.5;fill:none"/>
   </svg:g>
 </xsl:if>
 <!-- Изменяем порядок данных, чтобы компенсировать поворот. -->
 <!-- См. сортировку ниже -->
  <xsl:variable name="data-order">
    <xsl:choose>
      <xsl:when test="$orientation mod 360 >= 180">ascending</xsl:when>
      <xsl:otherwise>descending</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <svg:g transform="translate({$offsetX},{$offsetY})
```

```
                translate({$width div 2},{$height div 2})
                rotate({$orientation - 180})
                translate({-$width div 2},{-$height div 2})
                scale(1,{$height div $max})">

  <xsl:for-each select="$data">
    <!-- Сортируем по позиции, чтобы можно было обходить данные -->
    <!-- в противоположном направлении, когда это требуется. -->
    <xsl:sort select="position(  )" data-type="number"
         order="{$data-order}"/>

    <xsl:variable name="pos" select="position(  )"/>
```

Ниже столбцы рисуются в виде отрезков прямых. Для изменения цвета и толщины линии следует переопределить шаблон `BarStyle`. Если бы мы использовали для рисования столбцов прямоугольники, то можно было бы дополнительно варьировать стиль рамки:

```
<svg:line x1="{$spacing * $pos}"
     y1="0"
     x2="{$spacing * $pos}"
     y2="{current(  )}" id="{$context}_bar_{$pos}">
   <xsl:attribute name="style">
     <xsl:value-of select="concat('stroke-width: ',$barWidth,'; ')"/>
     <xsl:call-template name="svgu:barStyle">
       <xsl:with-param name="pos" select="$pos"/>
       <xsl:with-param name="context" select="$context"/>
     </xsl:call-template>
   </xsl:attribute>
</svg:line>

<!-- Если пользователю потребуются надписи, мы разместим текст над -->
<!-- столбцом. Чтобы текст выводился правильно, несмотря на поворот -->
<!-- и масштабирование системы координат, необходима сложная -->
<!-- последовательность преобразований. -->
<xsl:if test="$barLabel">
   <svg:text x="{$spacing * $pos}"
        y="{current(  ) * ($height div $max)}"
        transform="scale(1,{$max div $height})
                   translate(0,10)
                   translate({$spacing * $pos},{current(  ) *
                             ($height div $max)})
                   rotate({180 - $orientation})
                   translate({-$spacing * $pos},
                    {-current(  ) * ($height div $max)})"
```

```
          id="{$context}_barLabel_{$pos}">
          <xsl:attribute name="style">
             <xsl:call-template name="svgu:barLabelStyle">
                 <xsl:with-param name="pos" select="$pos"/>
                 <xsl:with-param name="context" select="$context"/>
             </xsl:call-template>
           </xsl:attribute>
           <xsl:value-of select="."/>
         </svg:text>
       </xsl:if>
     </xsl:for-each>
  </svg:g>

</xsl:template>

<xsl:template name="svgu:barStyle">
  <xsl:param name="pos"/>
  <xsl:param name="context"/>
  <xsl:variable name="colors" select="document('')/*/svgu:color"/>
   <xsl:value-of
        select="concat('stroke: ',$colors[($pos - 1 ) mod count($colors)
                + 1])"/>
</xsl:template>

<xsl:template name="svgu:barLabelStyle">
  <xsl:param name="pos"/>
  <xsl:param name="context"/>
  <xsl:value-of select=" 'text-anchor: middle' "/>
</xsl:template>
```

Следующая таблица стилей строит столбчатую диаграмму по переданным ей данным. Результаты показаны на рис. 11.8.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
  exclude-result-prefixes="svgu">

<xsl:import href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
  doctype-public="-//W3C//DTD SVG 1.0/EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>
```

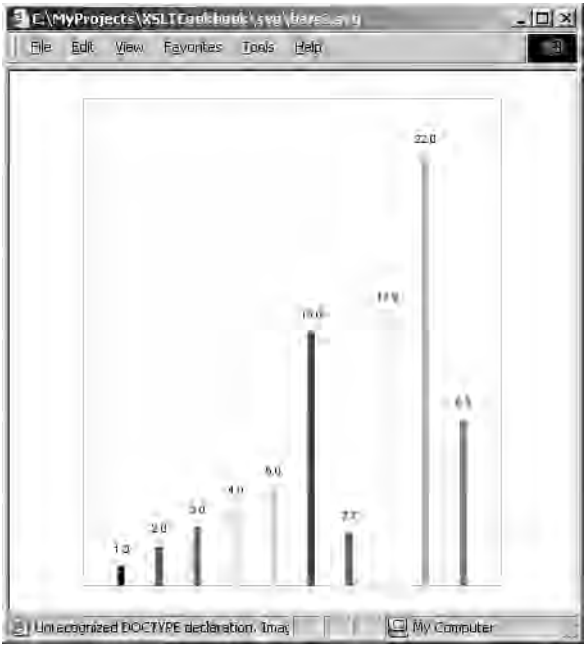


Рис. 11.8. Сгенерированная столбчатая диаграмма

```
<test:data>1.0</test:data>
<test:data>2.0</test:data>
<test:data>3.0</test:data>
<test:data>4.0</test:data>
<test:data>5.0</test:data>
<test:data>13.0</test:data>
<test:data>2.7</test:data>
<test:data>13.9</test:data>
<test:data>22.0</test:data>
<test:data>8.5</test:data>

<xsl:template match="/">

<svg:svg width="400" height="400">

  <xsl:call-template name="svgu:bars">
     <xsl:with-param name="data" select="document('')/*/test:data"/>
     <xsl:with-param name="width" select=" '300' "/>
     <xsl:with-param name="height" select=" '350' "/>
     <xsl:with-param name="orientation" select=" '0' "/>
     <xsl:with-param name="offsetX" select=" '50' "/>
     <xsl:with-param name="offsetY" select=" '25' "/>
```

```
      <xsl:with-param name="boundingBox" select="1"/>
      <xsl:with-param name="barLabel" select="1"/>
      <xsl:with-param name="max" select="25"/>
    </xsl:call-template>

</svg:svg>

</xsl:template>

<xsl:template name="svgu:barLabelStyle">
  <xsl:param name="pos"/>
  <xsl:param name="context"/>
  <xsl:text>text-anchor: middle; font-size: 8</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

А ниже показано, как создать горизонтальную диаграмму (рис. 11.9). В коде нет ограничений на угол поворота, хотя разумно, конечно, использовать только углы, кратные 90 градусам[1].

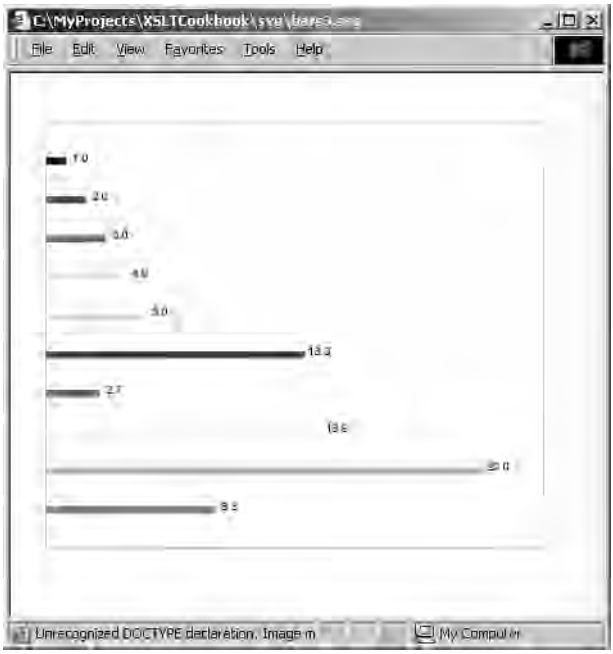


Рис. 11.9. Повернутая столбчатая диаграмма

[1] Если хотите позлить доставшего вас начальника, можете сгенерировать диаграмму с углом наклона 72 градуса. А еще лучше задайте угол 1 градус и попросите коллег поклясться, что диаграмма абсолютно ровная!

```
<xsl:call-template name="svgu:bars">
    <xsl:with-param name="data" select="document('')/*/test:data"/>
    <xsl:with-param name="width" select=" '300' "/>
    <xsl:with-param name="height" select=" '350' "/>
    <xsl:with-param name="orientation" select=" '90' "/>
    <xsl:with-param name="offsetX" select=" '50' "/>
    <xsl:with-param name="offsetY" select=" '25' "/>
    <xsl:with-param name="boundingBox" select="1"/>
    <xsl:with-param name="barLabel" select="1"/>
    <xsl:with-param name="max" select="25"/>
</xsl:call-template>
```

Линейные графики

Оси и координатные сетки бесполезны, если нельзя вывести данные. Один из самых распространенных способов представления данных – линейный график, на котором одно значение показано как функция другого. Можно написать утилиту, которая обрабатывает один набор данных, и вызвать ее несколько раз для рисования различных графиков в одних и тех же осях:

```
<xsl:template name="svgu:xyPlot">
  <xsl:param name="dataX" select="/.."/> <!-- значения x -->
  <xsl:param name="dataY" select="/.."/>
  <xsl:param name="offsetX" select="0"/>
  <xsl:param name="offsetY" select="0"/>
  <xsl:param name="width" select="500"/>
  <xsl:param name="height" select="500"/>
  <xsl:param name="boundingBox" select="false(  )"/>
  <xsl:param name="context"/>
  <xsl:param name="maxX">
     <xsl:call-template name="emath:max">
        <xsl:with-param name="nodes" select="$dataX"/>
     </xsl:call-template>
  </xsl:param>
  <xsl:param name="maxY">
     <xsl:call-template name="emath:max">
        <xsl:with-param name="nodes" select="$dataY"/>
     </xsl:call-template>
  </xsl:param>

  <xsl:variable name="scaleX" select="$width div $maxX"/>
  <xsl:variable name="scaleY" select="$height div $maxY"/>
```

В этом разделе мы для простоты воспользовались написанной на Java функцией расширения, но можно было бы реализовать функцию `max` и на XPath:

```
select="($scaleX > $scaleY) * $scaleX + not($scaleX > $scaleY)
* $scaleY)":

    <xsl:variable name="scale" select="Math:max($scaleX,$scaleY)"/>

    <xsl:if test="$boundingBox">
       <svg:g transform="translate({$offsetX},{$offsetY})">
          <svg:rect  x="0" y="0" height="{$height}" width="{$width}"
                     style="stroke: black;stroke-width:0.5;
                            stroke-opacity:0.5;fill:none"/>
       </svg:g>
    </xsl:if>
```

Я представляю кривую в виде набора прямолинейных отрезков, но, применив кривые Безье, можно было бы сгладить ее, правда, ценой усложнения кода. Поскольку эта книга все-таки посвящена XSLT, а не SVG, то не будем жертвовать простотой. Для использования кубической кривой Безье можно было бы рисовать сразу три точки, выбрав центральную в качестве управляющей, но я эту идею не проверял:

```
    <svg:path transform="translate({$offsetX},{$height + $offsetY})
                         scale({$scaleX},{-$scaleY})">
    <xsl:attribute name="d">
          <xsl:for-each select="$dataX">
          <xsl:variable name="pos" select="position(  )"/>
          <xsl:variable name="x" select="current(  ) "/>
          <xsl:variable name="y" select="$dataY[$pos]"/>
          <xsl:choose>
             <xsl:when test="$pos = 1">
                <xsl:text>M </xsl:text>
             </xsl:when>
             <xsl:otherwise> L </xsl:otherwise>
          </xsl:choose>
          <xsl:value-of select="$x"/>,<xsl:value-of select="$y"/>
       </xsl:for-each>
       </xsl:attribute>
       <xsl:attribute name="style">
          <xsl:call-template name="svgu:xyPlotStyle">
             <xsl:with-param name="scale" select="$scale"/>
             <xsl:with-param name="context" select="$context"/>
          </xsl:call-template>
       </xsl:attribute>
    </svg:path>
 </xsl:template>

 <xsl:template name="svgu:xyPlotStyle">
```

```
    <xsl:param name="context"/>
    <xsl:param name="scale"/>
    <xsl:value-of select="concat('fill: none; stroke: black; stroke-
          width:',1 div $scale,'; ')"/>
</xsl:template>
```

В следующей таблице стилей тестируется шаблон для вывода линейного графика. Полученные результаты показаны на рис. 11.10. Для простоты я поместил данные прямо в таблицу. В реальном приложении они должны извлекаться из другого XML-документа:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
  exclude-result-prefixes="svgu test">

<xsl:import href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
  doctype-public="-//W3C//DTD SVG 1.0/EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<test:xdata>0</test:xdata>
```

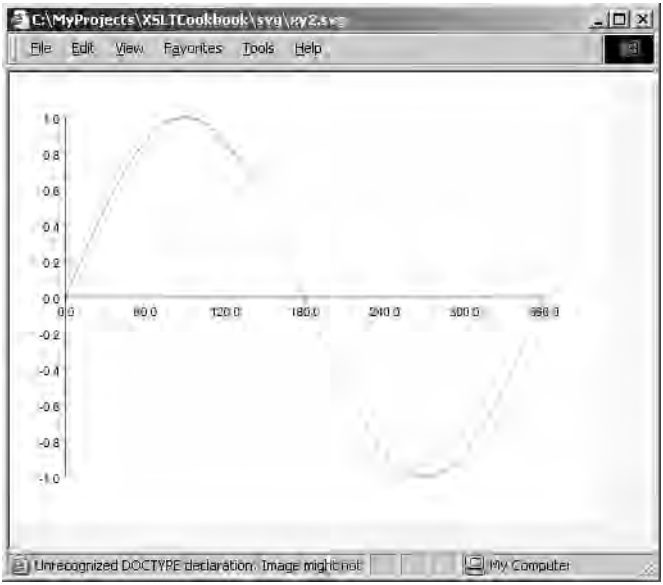


Рис. 11.10. Вывод линейного графика с помощью SVG и XSLT

```
<test:xdata>5</test:xdata>
<test:xdata>10</test:xdata>
<test:xdata>15</test:xdata>
<test:xdata>20</test:xdata>
<test:xdata>25</test:xdata>
<test:xdata>30</test:xdata>
<!-- Остальные координаты x опущены ... -->

<test:ydata>0</test:ydata>
<test:ydata>0.087155743</test:ydata>
<test:ydata>0.173648178</test:ydata>
<test:ydata>0.258819045</test:ydata>
<test:ydata>0.342020143</test:ydata>
<test:ydata>0.422618262</test:ydata>
<test:ydata>0.5</test:ydata>
<!-- Остальные координаты y опущены ... -->

<xsl:variable name="w" select="400"/>
<xsl:variable name="h" select="300"/>
<xsl:variable name="pwidth" select="$w * 0.8"/>
<xsl:variable name="pheight" select="$h * 0.8"/>
<xsl:variable name="offsetX" select="($w - $pwidth) div 2"/>
<xsl:variable name="offsetY" select="($h - $pheight) div 2"/>

<xsl:template match="/">

<svg:svg width="{$w}" height="{$h}">
  <xsl:call-template name="svgu:xyPlot">
     <xsl:with-param name="dataX" select="document('')/*/test:xdata"/>
     <xsl:with-param name="dataY" select="document('')/*/test:ydata"/>
     <xsl:with-param name="offsetX" select="$offsetX"/>
     <xsl:with-param name="offsetY" select="$offsetY"/>
     <xsl:with-param name="width" select="$pwidth"/>
     <xsl:with-param name="height" select="$pheight"/>
     <!--
     <xsl:with-param name="minY" select="-1"/>
     <xsl:with-param name="maxY" select="1"/>
     -->
  </xsl:call-template>

  <xsl:call-template name="svgu:xAxis">
     <xsl:with-param name="min" select="0"/>
     <xsl:with-param name="max" select="360"/>
     <xsl:with-param name="offsetX" select="$offsetX"/>
     <xsl:with-param name="offsetY" select="-$pheight div 2 + $offsetY"/>
```

```
        <xsl:with-param name="width" select="$pwidth"/>
        <xsl:with-param name="height" select="$pheight"/>
        <xsl:with-param name="majorTicks" select="6"/>
           <!-- Количество крупных делений по оси x -->
        <xsl:with-param name="minorTicks" select="4"/>
           <!-- Количество мелких делений по оси x -->
     </xsl:call-template>

     <xsl:call-template name="svgu:yAxis">
        <xsl:with-param name="min" select="-1"/>
        <xsl:with-param name="max" select="1"/>
        <xsl:with-param name="offsetX" select="$offsetX"/>
        <xsl:with-param name="offsetY" select="$offsetY"/>
        <xsl:with-param name="width" select="$pwidth"/>
        <xsl:with-param name="height" select="$pheight"/>
     </xsl:call-template>
  </svg:svg>

  </xsl:template>

  </xsl:stylesheet>
```

Следующая программа, результаты работы которой показаны на рис. 11.11, демонстрирует вывод нескольких графиков для разных наборов данных и изменения стиля линии за счет переопределения шаблона:

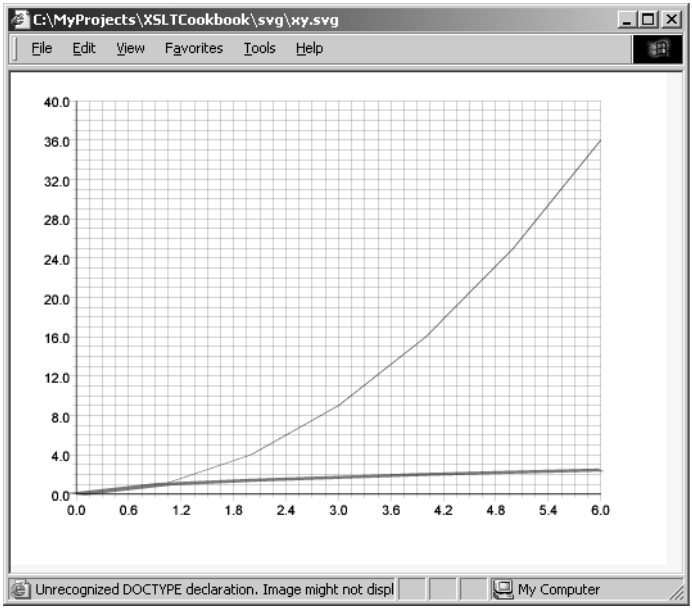


Рис. 11.11. Несколько графиков, сгенерированных с помощью XSLT

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
  exclude-result-prefixes="svgu test">

<xsl:import href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
  doctype-public="-//W3C//DTD SVG 1.0/EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<!-- Значения данных опущены ... -->

<xsl:variable name="w" select="400"/>
<xsl:variable name="h" select="300"/>
<xsl:variable name="pwidth" select="$w * 0.8"/>
<xsl:variable name="pheight" select="$h * 0.8"/>
<xsl:variable name="offsetX" select="($w - $pwidth) div 2"/>
<xsl:variable name="offsetY" select="($h - $pheight) div 2"/>

<xsl:template match="/">

<svg:svg width="{$w}" height="{$h}">

  <xsl:call-template name="svgu:xyPlot">
    <xsl:with-param name="dataX" select="document('')/*/test:xdata"/>
    <xsl:with-param name="dataY" select="document('')/*/test:ydata"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="maxY" select="40"/>
  </xsl:call-template>

  <xsl:call-template name="svgu:xyPlot">
    <xsl:with-param name="dataX" select="document('')/*/test:xdata"/>
    <xsl:with-param name="dataY" select="document('')/*/test:y2data"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="maxY" select="40"/>
```

```
    <xsl:with-param name="context" select="2"/>
  </xsl:call-template>

  <xsl:call-template name="svgu:xAxis">
    <xsl:with-param name="min" select="0"/>
    <xsl:with-param name="max" select="6"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="majorTopExtent" select="$pheight"/>
    <xsl:with-param name="minorTopExtent" select="$pheight"/>
  </xsl:call-template>

  <xsl:call-template name="svgu:yAxis">
    <xsl:with-param name="min" select="0"/>
    <xsl:with-param name="max" select="40"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="majorRightExtent" select="$pwidth"/>
    <xsl:with-param name="minorRightExtent" select="$pwidth"/>
  </xsl:call-template>

</svg:svg>

</xsl:template>

<!-- При определении специального стиля используется контекст, -->
<!-- позволяющий узнать, какая линия рисуется -->
<xsl:template name="svgu:xyPlotStyle">
   <xsl:param name="context"/>
   <xsl:param name="scale"/>
   <xsl:choose>
    <xsl:when test="$context = 2">
     <xsl:value-of select="concat('fill: none; stroke: red;
       stroke-width:',16 div $scale,'; ')"/>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="concat('fill: none; stroke: black;
        stroke-width:',1 div $scale,'; ')"/>
    </xsl:otherwise>
  </xsl:choose>
```

```
</xsl:template>

</xsl:stylesheet>
```

Генерация секторных диаграмм

Еще один распространенный способ визуального сравнения данных – секторные диаграммы. Можно написать утилиту, которая будет их генерировать. Для создания секторной диаграммы нужно иметь средства для создания одного сектора, а это неизбежно заводит нас в дебри тригонометрии. Поскольку в XSLT нет тригонометрических функций, придется воспользоваться расширением на языке Java. Конечно, это сразу же ограничивает переносимость таблицы стилей. Если без переносимости никак нельзя, можно реализовать синус и косинус на XSLT (см. указания в рецепте 3.5). А если переносимостью можно пожертвовать, то включите в таблицу показанный ниже код, где запрашиваются математические расширения на Java. Точный вид зависит от процессора, в главе 12 приведена дополнительная информация по этому поводу. Этот пример ориентирован на процессор Saxon:

```
<xsl:stylesheet
  <!-- версия 1.1 объявлена не действующей, но Saxon поддерживает ее -->
  <!-- ради xsl:script. -->
  version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:emath="http://www.exslt.org/math"
  xmlns:Math="java:java.lang.Math" extension-element-prefixes="Math"
        exclude-result-prefixes="svgu">

  <xsl:script implements-prefix="Math"
              xmlns:Math="java:java.lang.Math"
              language="java"
              src="java:java.lang.Math"/>

  <!-- Применяем XSLT-код, разработанный ранее в главе 2 -->
  <xsl:include href="../math/math.max.xslt"/>
  <xsl:include href="../math/math.min.xslt"/>
...
</xsl:stylesheet>
```

В шаблоне `svgu:pieSlice` реализована большая часть необходимой математики. Код основан на Perl-программе, приведенной в книге J. David Eisenberg *SVG Essentials* (O'Reilly, 2002). Рассказывать об основах тригонометрии в этой книге мы не будем, но по существу код позволяет нарисовать дуги (с помощью поворотов вокруг начала координат), компенсируя интуитивно не очевидную спецификацию дуг в SVG:

```
<xsl:variable name="svgu:pi" select="3.1415927"/>

<xsl:template name="svgu:pieSlice">
  <xsl:param name="cx" select="100"/>  <!-- Абсцисса центра -->
  <xsl:param name="cy" select="100"/>  <!-- Ордината центра -->
  <xsl:param name="r" select="50"/>    <!-- Радиус -->
  <xsl:param name="theta" select="0"/> <!-- Начальный угол в градусах -->
  <xsl:param name="delta" select="90"/>  <!-- Угловая длина дуги
                                          в градусах -->
  <xsl:param name="phi" select="0"/>  <!-- Угол поворота оси x -->
  <xsl:param name="style" select=" 'fill: red;' "/>
  <xsl:param name="num"/>
  <xsl:param name="context"/>

  <!-- Преобразуем градусы в радианы -->
  <xsl:variable name="theta1"
                select="$theta * $svgu:pi div 180"/>
  <xsl:variable name="theta2"
                select="($delta + $theta) * $svgu:pi div 180"/>
  <xsl:variable name="phi_r" select="$phi * $svgu:pi div 180"/>

  <!-- Находим координаты начала и конца дуги -->
  <xsl:variable name="x0"
     select="$cx +  Math:cos($phi_r) * $r * Math:cos($theta1) +
                    Math:sin(-$phi_r) * $r * Math:sin($theta1)"/>
  <xsl:variable name="y0"
     select="$cy + Math:sin($phi_r) * $r * Math:cos($theta1) +
                    Math:cos($phi_r) * $r * Math:sin($theta1)"/>

  <xsl:variable name="x1"
     select="$cx + Math:cos($phi_r) * $r * Math:cos($theta2) +
                    Math:sin(-$phi_r) * $r * Math:sin($theta2)"/>
  <xsl:variable name="y1"
      select="$cy + Math:sin($phi_r) * $r * Math:cos($theta2) +
                    Math:cos($phi_r) * $r * Math:sin($theta2)"/>

  <xsl:variable name="large-arc" select="($delta > 180) * 1"/>
  <xsl:variable name="sweep" select="($delta > 0) * 1"/>

  <svg:path style="{$style} id="{$context}_pieSlice_{$num}">
     <xsl:attribute name="d">
     <xsl:value-of select="concat('M ', $x0,' ',$y0,
                                  ' A ', $r,' ',$r,',',
                                  $phi,',',
```

```
                                    $large-arc,',',
                                    $sweep,',',
                                    $x1,' ',$y1,
                                    ' L ',$cx,' ',$cy,
                                    ' L ', $x0,' ',$y0)"/>

      </xsl:attribute>
   </svg:path>
</xsl:template>

<xsl:template name="svgu:pieSliceLabel">
  <xsl:param name="label" />          <!-- Надпись -->
  <xsl:param name="cx" select="100"/> <!-- Абсцисса центра -->
  <xsl:param name="cy" select="100"/> <!-- Ордината центра -->
  <xsl:param name="r" select="50"/>   <!-- Радиус -->
  <xsl:param name="theta" select="0"/> <!-- Начальный угол в градусах -->
  <xsl:param name="delta" select="90"/> <!-- Угловая длина дуги в градусах -->
  <xsl:param name="style" select=" 'font-size: 18;' "/>
  <xsl:param name="num"/>
  <xsl:param name="context"/>

  <!-- Преобразуем градусы в радианы -->
  <xsl:variable  name="theta2"
                 select="(($delta + $theta) mod 360 + 360) mod 360"/>
  <!-- Нормализуем углы -->
  <xsl:variable name="theta2_r" select="$theta2 * $svgu:pi div 180"/>
  <xsl:variable name="x" select="$cx + $r * Math:cos($theta2_r)"/>
  <xsl:variable name="y" select="$cy + $r * Math:sin($theta2_r)"/>

  <!-- Вычисляем координаты начала текста с учетом положения сектора -->
  <!-- в круге. Наша цель - расположить текст более-менее равномерно. -->
  <xsl:variable name="anchor">
    <xsl:choose>
      <xsl:when test="contains($style,'text-anchor')"></xsl:when>
      <xsl:when test="$theta2 >= 0 and $theta2 &lt;= 45">start</xsl:when>
      <xsl:when test="$theta2 > 45 and
                      $theta2 &lt;= 135">middle</xsl:when>
      <xsl:when test="$theta2 > 135 and $theta2 &lt;= 225">end</xsl:when>
      <xsl:when test="$theta2 > 225 and
                      $theta2 &lt;= 315">middle</xsl:when>
      <xsl:otherwise>start</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>

  <svg:text x="{$x}" y="{$y}"
```

```
            style="text-anchor:{$anchor};{$style}"
            id="{$context}_pieSliceLabel_{$num}">
        <xsl:value-of select="$label"/>
    </svg:text>
</xsl:template>

<xsl:template name="svgu:pie">
    <xsl:param name="data" select="/.."/> <!-- Данные для диаграммы -->
    <xsl:param name="cx" select="100"/>  <!-- Абсцисса центра -->
    <xsl:param name="cy" select="100"/>  <!-- Ордината центра -->
    <xsl:param name="r" select="50"/>    <!-- Радиус -->
    <xsl:param name="theta" select="-90"/>  <!-- Начальный угол первого
                                                сектора в градусах -->
    <xsl:param name="context"/>       <!-- Пользовательские данные для
                                          идентификации этого вызова -->

    <xsl:call-template name="svgu:pieImpl">
        <xsl:with-param name="data" select="$data"/>
        <xsl:with-param name="cx" select="$cx"/>
        <xsl:with-param name="cy" select="$cy"/>
        <xsl:with-param name="r" select="$r"/>
        <xsl:with-param name="theta" select="$theta"/>
        <xsl:with-param name="sum" select="sum($data)"/>
        <xsl:with-param name="context" select="$context"/>
    </xsl:call-template>

</xsl:template>

<!-- Рекурсивная реализация -->
<xsl:template name="svgu:pieImpl">
    <xsl:param name="data" />
    <xsl:param name="cx" />
    <xsl:param name="cy" />
    <xsl:param name="r" />
    <xsl:param name="theta"/>
    <xsl:param name="sum"/>
    <xsl:param name="context"/>
    <xsl:param name="i" select="1"/>

    <xsl:if test="count($data) >= $i">
        <xsl:variable name="delta" select="($data[$i] * 360) div $sum"/>

        <!-- Рисуем сектор -->
        <xsl:call-template name="svgu:pieSlice">
```

```
            <xsl:with-param name="cx" select="$cx"/>
            <xsl:with-param name="cy" select="$cy"/>
            <xsl:with-param name="r" select="$r"/>
            <xsl:with-param name="theta" select="$theta"/>
            <xsl:with-param name="delta" select="$delta"/>
            <xsl:with-param name="style">
               <xsl:call-template name="svgu:pieSliceStyle">
                  <xsl:with-param name="i" select="$i"/>
                  <xsl:with-param name="context" select="$context"/>
               </xsl:call-template>
            </xsl:with-param>
            <xsl:with-param name="num" select="$i"/>
            <xsl:with-param name="context" select="$context"/>
         </xsl:call-template>

         <!-- Рекурсивный вызов для рисования следующего сектора -->
         <xsl:call-template name="svgu:pieImpl">
            <xsl:with-param name="data" select="$data"/>
            <xsl:with-param name="cx" select="$cx"/>
            <xsl:with-param name="cy" select="$cy"/>
            <xsl:with-param name="r" select="$r"/>
            <xsl:with-param name="theta" select="$theta + $delta"/>
            <xsl:with-param name="sum" select="$sum"/>
            <xsl:with-param name="context" select="$context"/>
            <xsl:with-param name="i" select="$i + 1"/>
         </xsl:call-template>
      </xsl:if>

   </xsl:template>

   <!-- Расставляем надписи для каждого сектора -->
   <xsl:template name="svgu:pieLabels">
         <xsl:param name="data" select="/.."/>  <!-- Данные для секторов -->
         <xsl:param name="labels" select="$data"/>  <!-- Набор узлов,
               соответствующих надписям на диаграмме. По умолчанию data -->
         <xsl:param name="cx" select="100"/>    <!-- Абсцисса центра -->
         <xsl:param name="cy" select="100"/>    <!-- Ордината центра -->
         <xsl:param name="r" select="50"/>    <!-- Радиус -->
       <xsl:param name="theta" select="-90"/> <!-- Начальный угол первого
                                                   сектора в градусах -->
       <xsl:param name="context"/>   <!-- Пользовательские данные для
                                          идентификации этого вызова -->

         <xsl:call-template name="svgu:pieLabelsImpl">
```

```
            <xsl:with-param name="data" select="$data"/>
            <xsl:with-param name="labels" select="$labels"/>
            <xsl:with-param name="cx" select="$cx"/>
            <xsl:with-param name="cy" select="$cy"/>
            <xsl:with-param name="r" select="$r"/>
            <xsl:with-param name="theta" select="$theta"/>
            <xsl:with-param name="sum" select="sum($data)"/>
            <xsl:with-param name="context" select="$context"/>
         </xsl:call-template>

</xsl:template>

<xsl:template name="svgu:pieLabelsImpl">
   <xsl:param name="data" />
   <xsl:param name="labels"/>
   <xsl:param name="cx" />
   <xsl:param name="cy" />
   <xsl:param name="r" />
   <xsl:param name="theta"/>
   <xsl:param name="sum"/>
   <xsl:param name="context"/>
   <xsl:param name="i" select="1"/>

   <xsl:if test="count($data) >= $i">
      <xsl:variable name="delta" select="($data[$i] * 360) div $sum"/>

      <!-- Выводим надпись для сектора -->
      <xsl:call-template name="svgu:pieSliceLabel">
      <xsl:with-param name="label" select="$labels[$i]"/>
      <xsl:with-param name="cx" select="$cx"/>
      <xsl:with-param name="cy" select="$cy"/>
      <xsl:with-param name="r" select="$r"/>
      <xsl:with-param name="theta" select="$theta"/>
      <xsl:with-param name="delta" select="$delta div 2"/>
      <xsl:with-param name="style">
         <xsl:call-template name="svgu:pieSliceLabelStyle">
         <xsl:with-param name="i" select="$i"/>
         <xsl:with-param name="value" select="$data[$i]"/>
         <xsl:with-param name="label" select="$labels[$i]"/>
         <xsl:with-param name="context" select="$context"/>
         </xsl:call-template>
      </xsl:with-param>
      <xsl:with-param name="num" select="$i"/>
      <xsl:with-param name="context" select="$context"/>
```

```
      </xsl:call-template>

      <!-- Рекурсивный вызов для вывода следующей надписи -->
      <xsl:call-template name="svgu:pieLabelsImpl">
       <xsl:with-param name="data" select="$data"/>
       <xsl:with-param name="labels" select="$labels"/>
       <xsl:with-param name="cx" select="$cx"/>
       <xsl:with-param name="cy" select="$cy"/>
       <xsl:with-param name="r" select="$r"/>
       <xsl:with-param name="theta" select="$theta + $delta"/>
       <xsl:with-param name="sum" select="$sum"/>
       <xsl:with-param name="context" select="$context"/>
       <xsl:with-param name="i" select="$i + 1"/>
      </xsl:call-template>
    </xsl:if>

  </xsl:template>

  <!-- Переопределить для изменения стиля сектора -->
  <xsl:template name="svgu:pieSliceStyle">
    <xsl:param name="i"/>
    <xsl:param name="context"/>
    <xsl:variable name="colors" select="document('')/*/svgu:color"/>
    <xsl:value-of select="concat('stroke:black;
                                  stroke-width:0.5;
                                  fill: ',$colors[($i - 1 ) mod
                                            count($colors) + 1])"/>
  </xsl:template>

  <!-- Переопределить для изменения стиля надписи -->
  <xsl:template name="svgu:pieSliceLabelStyle">
    <xsl:param name="i"/>
    <xsl:param name="value"/>
    <xsl:param name="label" />
    <xsl:param name="context"/>
    <xsl:text>font-size: 16;</xsl:text>
  </xsl:template>
```

Следующая таблица стилей создает секторную диаграмму, показанную на рис. 11.12:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
```

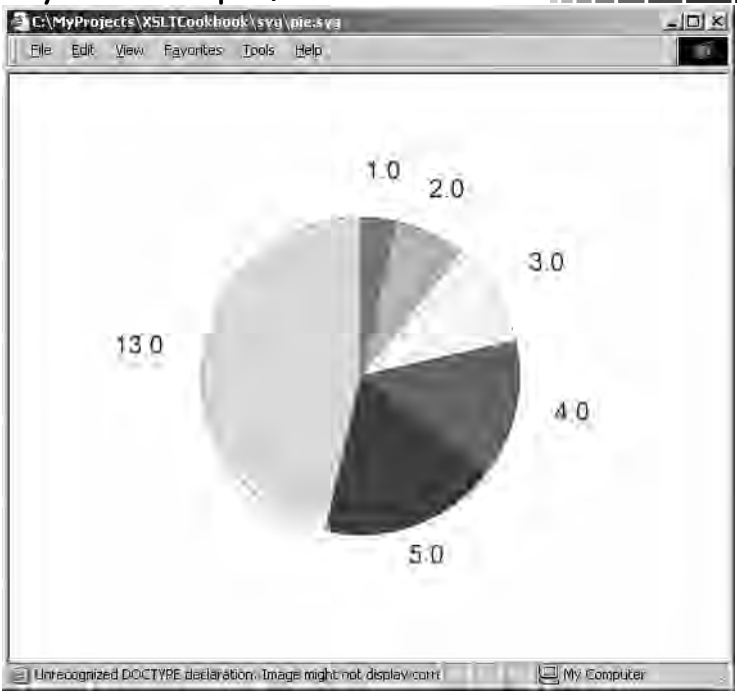


Рис. 11.12. Сгенерированная секторная диаграмма

```
   xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
   exclude-result-prefixes="svgu test">

<xsl:include href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
   doctype-public="-//W3C//DTD SVG 1.0/EN"
   doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<test:data>1.0</test:data>
<test:data>2.0</test:data>
<test:data>3.0</test:data>
<test:data>4.0</test:data>
<test:data>5.0</test:data>
<test:data>13.0</test:data>

<xsl:template match="/">

<svg:svg width="500" height="500">
   <xsl:call-template name="svgu:pie">
      <xsl:with-param name="data" select="document('')/*/test:data"/>
      <xsl:with-param name="cx" select="250"/>
```

```
      <xsl:with-param name="cy" select="250"/>
      <xsl:with-param name="r" select="100"/>
      <xsl:with-param name="theta" select="-90"/>
   </xsl:call-template>

   <xsl:call-template name="svgu:pieLabels">
      <xsl:with-param name="data" select="document('')/*/test:data"/>
      <xsl:with-param name="cx" select="250"/>
      <xsl:with-param name="cy" select="250"/>
      <xsl:with-param name="r" select="125"/>
      <xsl:with-param name="theta" select="-90"/>
   </xsl:call-template>

</svg:svg>

</xsl:template>
```

Графики вида «начало-максимум-минимум-конец»

Такие графики (Open-Hi-Lo-Close) обычно используются для представления данных о торгах ценными бумагами, но применимы и в других случаях (например, чтобы показать минимум, максимум, среднее и медиану). В шаблон данные передаются в виде четырех разных наборов узлов – по одному для каждой последовательности значений. Обязательны только наборы максимальных и минимальных значений. Шаблон умеет обрабатывать последовательности с отсутствующими данными.

```
<xsl:template name="svgu:openHiLoClose">
   <xsl:param name="openData" select="/.."/>
   <xsl:param name="hiData" select="/.."/>
   <xsl:param name="loData" select="/.."/>
   <xsl:param name="closeData" select="/.."/>
   <xsl:param name="width" select=" '500' "/>
   <xsl:param name="height" select=" '500' "/>
   <xsl:param name="offsetX" select="0"/>
   <xsl:param name="offsetY" select="0"/>
   <xsl:param name="openCloseExtent" select="8"/>
   <xsl:param name="max">
      <xsl:call-template name="emath:max">
         <xsl:with-param name="nodes" select="$hiData"/>
      </xsl:call-template>
    </xsl:param>
   <xsl:param name="min">
      <xsl:call-template name="emath:min">
         <xsl:with-param name="nodes" select="$loData"/>
      </xsl:call-template>
   </xsl:param>
```

```
<xsl:param name="context"/>

<xsl:variable name="hiCount" select="count($hiData)"/>
<xsl:variable name="loCount" select="count($loData)"/>
<xsl:variable name="openCount" select="count($openData)"/>
<xsl:variable name="closeCount" select="count($closeData)"/>

<xsl:variable name="numBars" select="Math:min($hiCount, $loCount)"/>

<xsl:variable name="spacing" select="$width div ($numBars + 1)"/>

<xsl:variable name="range" select="$max - $min"/>
<xsl:variable name="scale" select="$height div $range"/>

<svg:g transform="translate({$offsetX},{$offsetY+$height})
                      scale(1,{-$scale})
                      translate(0,{-$min})">

<xsl:for-each select="$hiData">
   <xsl:variable name="pos" select="position(  )"/>

   <!-- рисуем линию максимум-минимум -->
   <svg:line x1="{$spacing * $pos}"
             y1="{$loData[$pos]}"
             x2="{$spacing * $pos}"
             y2="{current(  )}"id="{$context}_highLow_{$pos}">
      <xsl:attribute name="style">
      <xsl:call-template name="svgu:hiLoBarStyle">
      <xsl:with-param name="pos" select="$pos"/>
      <xsl:with-param name="context" select="$context"/>
      </xsl:call-template>
      </xsl:attribute>
      </svg:line>

      <!-- Рисуем метку для данных на момент открытия, если таковые
      имеются -->
      <xsl:if test="$openCount >= $pos">
         <svg:line x1="{$spacing * $pos - $openCloseExtent}"
                y1="{$openData[$pos]}"
                x2="{$spacing * $pos}"
                y2="{$openData[$pos]}"
                id="{$context}_open_{$pos}">
         <xsl:attribute name="style">
            <xsl:call-template name="svgu:openCloseBarStyle">
```

```
                <xsl:with-param name="pos" select="$pos"/>
                <xsl:with-param name="scale" select="$scale"/>
                <xsl:with-param name="context" select="$context"/>
                </xsl:call-template>
            </xsl:attribute>
            </svg:line>
         </xsl:if>

         <!-- Рисуем метку для данных на момент открытия, если таковые
         имеются -->
         <xsl:if test="$closeCount >= $pos">
            <svg:line x1="{$spacing * $pos}"
                      y1="{$closeData[$pos]}"
                      x2="{$spacing * $pos +  $openCloseExtent}"
                      y2="{$closeData[$pos]}"
                      id="{$context}_close_{$pos}">
               <xsl:attribute name="style">
                  <xsl:call-template name="svgu:openCloseBarStyle">
                     <xsl:with-param name="pos" select="$pos"/>
                     <xsl:with-param name="scale" select="$scale"/>
                     <xsl:with-param name="context" select="$context"/>
                  </xsl:call-template>
               </xsl:attribute>
            </svg:line>
         </xsl:if>

      </xsl:for-each>
   </svg:g>

</xsl:template>

<xsl:template name="svgu:hiLoBarStyle">
   <xsl:param name="pos"/>
   <xsl:param name="context"/>
   <xsl:text>stroke: black; stroke-width: 1 </xsl:text>
</xsl:template>

<xsl:template name="svgu:openCloseBarStyle">
   <xsl:param name="pos"/>
   <xsl:param name="scale"/>
   <xsl:param name="context"/>
   <xsl:text>stroke: black; stroke-width: </xsl:text>
   <xsl:value-of select="2 div $scale"/>
</xsl:template>

</xl:stylesheet>
```

С помощью этих шаблонов можно вывести график, показанный на рис. 11.13:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
   exclude-result-prefixes="svgu">

<xsl:include href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
  doctype-public="-//W3C//DTD SVG 1.0/EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<xsl:template match="/">

<svg:svg width="600" height="400">

   <xsl:call-template name="svgu:openHiLoClose">
      <xsl:with-param name="openData" select="*/row/open"/>
      <xsl:with-param name="hiData" select="*/row/high"/>
      <xsl:with-param name="loData" select="*/row/low"/>
      <xsl:with-param name="closeData" select="*/row/close"/>
```

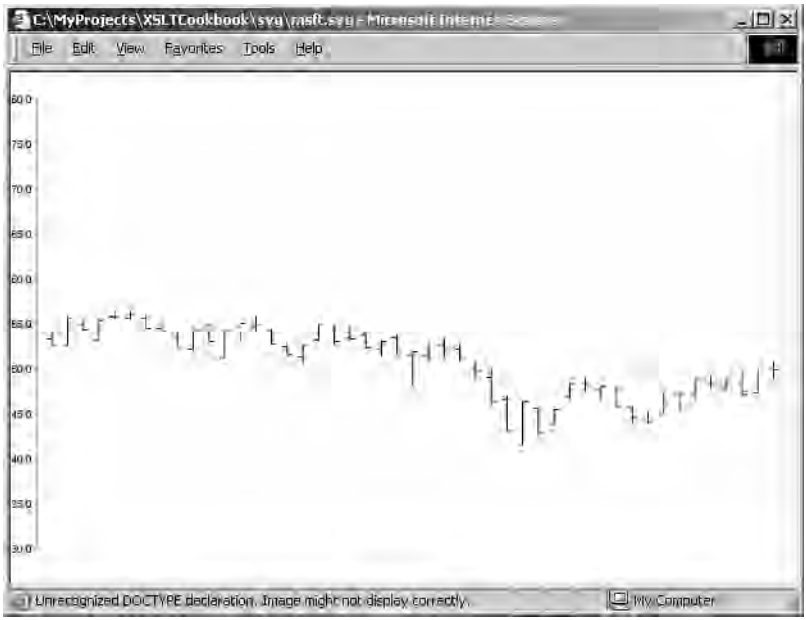


Рис. 11.13. График «начало-максимум-минимум-конец», сгенерированный с помощью XSLT

```
      <xsl:with-param name="min" select="30"/>
      <xsl:with-param name="max" select="80"/>
      <xsl:with-param name="width" select="600"/>
      <xsl:with-param name="height" select="350"/>
      <xsl:with-param name="offsetX" select="20"/>
      <xsl:with-param name="offsetY" select="20"/>
      <xsl:with-param name="boundingBox" select="1"/>
   </xsl:call-template>

   <xsl:call-template name="svgu:yAxis">
      <xsl:with-param name="min" select="30"/>
      <xsl:with-param name="max" select="80"/>
      <xsl:with-param name="offsetX" select="20"/>
      <xsl:with-param name="offsetY" select="20"/>
      <xsl:with-param name="width" select="600"/>
      <xsl:with-param name="height" select="350"/>
   </xsl:call-template>

</svg:svg>

</xsl:template>

</xsl:stylesheet>
```

XSLT 2.0

Для переноса этих рецептов с версии 1.0 на 2.0 нужны минимальные изменения. Поскольку у шаблонов довольно много параметров, то имеет смысл описать типы параметров и переменных, это уменьшит число ошибок во время выполнения. Параметры, описывающие размеры, например длина и ширина, должны иметь тип `xs:double`, а относящиеся к числу делений и т.п. – тип `xs:integer`. У некоторых параметров должен быть тип `xs:boolean`, но с ними все понятно, потому что они по умолчанию принимают значение `true( )` или `false()`.

Обсуждение

Преобразования XML в SVG обычно не тривиальны. Для графического представления данных даже в таких относительно простых случаях, как выше, требуется тщательное планирование. Начинать каждое преобразование с чистого листа было бы глупо – необходим комплект повторно используемых утилит. Я уделил внимание разработке таких утилит для построения диаграмм, а вы можете заняться инструментарием для других областей применения. Методика проектирования предполагает разбиение задачи на две части: создание готовых компонентов и шаблонов, в которых эти компоненты объединяются разными способами. Главное, чтобы каждый шаблон получал достаточно информации для преобразования системы

координат так, чтобы она была совместима с графикой, подготовленной независимыми компонентами. Например, у большинства представленных шаблонов есть параметры `$min` и `$max`, хотя разумные значения можно было бы вычислить по входным данным. Но наличие таких параметров позволяет вызывающей программе переопределить умолчания и рассматривать диапазон отображаемых на графике данных как единое целое.

При разработке шаблонов было принято решение получать информацию о стилях путем вызова шаблонов по умолчанию, которые импортирующая таблица может переопределить. Во многих случаях такую информацию можно было бы передать с помощью дополнительных параметров. Но нами выбран подход на основе обратных вызовов, поскольку он обеспечивает большую гибкость стилизации. Посмотрите, как можно было бы изменять внешний вид секторов или столбцов в зависимости от точки, наносимой на график:

```
<xsl:template name="svgu:pieSliceStyle">
  <xsl:param name="i"/>
  <xsl:param name="context"/>
  <xsl:variable name="colors" select="document('')/*/svgu:color"/>
  <xsl:value-of select="concat('stroke:black;
                                stroke-width:0.5;
                                fill: ',$colors[($i - 1 ) mod
                                          count($colors) + 1])"/>
</xsl:template>
```

Таким функциям можно было бы даже передавать дополнительные параметры из главного шаблона. Например, если передать само значение данных, то в зависимости от его величины можно было бы выбирать цвет. У такой техники, правда, есть ограничение: таблица стилей может переопределить любой шаблон не более одного раза. Чтобы как-то обойти это ограничение, мы передаем в качестве параметра определяемый пользователем контекст. В зависимости от его значения переопределенный шаблон может изменять свое поведение. У контекста есть и другое назначение – он может служить основой для генерации атрибута `id` в элементах SVG. Это полезно, если вы намереваетесь затем как-то взаимодействовать со сгенерированным SVG-документом (см. рецепт 11.4).

В последнем примере из этого рецепта мы создадим сложную диаграмму, в которой данные о колебаниях курса акций представлены графиком «открытие-максимум-минимум-закрытие», данные об объеме торгов – столбчатой диаграммой, а данные о среднем объеме – линейным графиком. При этом шкалы цен и объемов откладываются на разных осях ординат. Результаты представлены на рис. 11.14.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:emath="http://www.exslt.org/math"
```

Рис. 11.14. Сложная комбинация графиков

```
exclude-result-prefixes="svgu">

<xsl:include href="svg-utils.xslt"/>

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
 doctype-public="-//W3C//DTD SVG 1.0/EN"
 doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"/>

<xsl:variable name="width" select="600"/>
<xsl:variable name="height" select="500"/>
<xsl:variable name="pwidth" select="$width * 0.8"/>
<xsl:variable name="pheight" select="$height * 0.8"/>
<xsl:variable name="offsetX" select="($width - $pwidth) div 2"/>
<xsl:variable name="offsetY" select="10"/>

<xsl:variable name="dataMin">
  <xsl:call-template name="emath:min">
    <xsl:with-param name="nodes" select="//Low"/>
  </xsl:call-template>
```

```
</xsl:variable>

<xsl:variable name="dataMax">
  <xsl:call-template name="emath:max">
    <xsl:with-param name="nodes" select="//High"/>
  </xsl:call-template>
</xsl:variable>

<xsl:variable name="min" select="$dataMin * 0.9"/>
<xsl:variable name="max" select="$dataMax * 1.1"/>

<xsl:template match="/">

<svg:svg width="{$width}" height="{$height}">

<svg:text x="{$width div 2}" y="{2 * $offsetY}"
    style="text-anchor:middle; font-size:24">MSFT Stock Chart</svg:text>
<svg:text x="{$width div 2}" y="{4 * $offsetY}"
    style="text-anchor:middle; font-size:12">05/23/2002 to 08/16/2002
</svg:text>
<!-- ЦЕНА -->

  <xsl:call-template name="svgu:openHiLoClose">
    <xsl:with-param name="openData" select="*/row/Open"/>
    <xsl:with-param name="hiData" select="*/row/High"/>
    <xsl:with-param name="loData" select="*/row/Low"/>
    <xsl:with-param name="closeData" select="*/row/Close"/>
    <xsl:with-param name="min" select="$min"/>
    <xsl:with-param name="max" select="$max"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="boundingBox" select="1"/>
  </xsl:call-template>

  <xsl:call-template name="svgu:yAxis">
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$offsetY"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$pheight"/>
    <xsl:with-param name="min" select="$min"/>
    <xsl:with-param name="max" select="$max"/>
    <xsl:with-param name="context" select=" 'price' "/>
  </xsl:call-template>
```

```
  <!-- ОБЪЕМ -->
  <xsl:variable name="vheight" select="100"/>

  <xsl:call-template name="svgu:bars">
    <xsl:with-param name="data" select="*/row/Volume"/>
    <xsl:with-param name="width" select="$pwidth"/>
    <xsl:with-param name="height" select="$vheight"/>
    <xsl:with-param name="orientation" select="0"/>
    <xsl:with-param name="offsetX" select="$offsetX"/>
    <xsl:with-param name="offsetY" select="$pheight - $offsetY"/>
    <xsl:with-param name="barLabel" select="false(  )"/>
    <xsl:with-param name="min" select="0"/>
    <xsl:with-param name="max" select="1500000"/>
  </xsl:call-template>

  <!-- Чтобы линейный график начинался в точке расположения первого -->
  <!-- столбца и заканчивался там, где расположен последний -->
  <xsl:variable name="spacing" select="$pwidth div count(*/row/High) + 1"/>

  <xsl:call-template name="svgu:xyPlot">
    <xsl:with-param name="dataY" select="*/row/Vol10MA"/>
    <xsl:with-param name="width" select="$pwidth - 2 * $spacing"/>
    <xsl:with-param name="height" select="$vheight"/>
    <xsl:with-param name="offsetX" select="$offsetX + $spacing"/>
    <xsl:with-param name="offsetY" select="$pheight - $offsetY"/>
    <xsl:with-param name="minY" select="0"/>
    <xsl:with-param name="maxY" select="1500000"/>
  </xsl:call-template>

    <xsl:call-template name="svgu:yAxis">
      <xsl:with-param name="offsetX" select="$width - $offsetX"/>
      <xsl:with-param name="offsetY" select="$height - $vheight - $offsetY"/>
      <xsl:with-param name="width" select="$pwidth"/>
      <xsl:with-param name="height" select="$vheight"/>
      <xsl:with-param name="min" select="0"/>
      <xsl:with-param name="max" select="1500000"/>
      <xsl:with-param name="context" select=" 'volume' "/>
    </xsl:call-template>

  </svg:svg>

</xsl:template>

<xsl:template name="svgu:barStyle">
  <xsl:text>stroke: black; stroke-wdth: 0.15</xsl:text>
```

```
</xsl:template>

<xsl:template name="svgu:xyPlotStyle">
  <xsl:param name="context"/>
  <xsl:param name="scale"/>
  <xsl:value-of select="concat('fill: none; stroke: black; stroke-width:',4
div $scale,'; ')"/>
</xsl:template>

<xsl:template name="yAxisLabelStyle">
  <xsl:param name="context"/>
  <xsl:choose>
    <xsl:when test="$context = 'price'">
      <xsl:text>text-anchor:end;font-size:8;baseline-shift:-50%</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>text-anchor:start;font-size:8;baseline-shift:-50%</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Отодвинуть подписи на шкале объемов от делений -->
<xsl:template name="yAxisLabelXOffset">
  <xsl:param name="context"/>
  <xsl:if test="$context = 'volume'">
    <xsl:value-of select="6"/>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

11.3. Создание графического представления деревьев

Задача

Требуется представить иерархическую структуру данных в виде дерева.

Решение

В этом разделе мы рассмотрим два разных алгоритма рисования дерева. Есть и более сложные, но эти дают вполне приемлемые результаты.

Если бы дерево было сбалансированным, то нарисовать его было бы просто; надо лишь разделить имеющее горизонтальное пространство на количество узлов

на каждом уровне, а вертикальное – на число уровней[1]. К сожалению, реальные деревья не всегда симметричны. Поэтому алгоритм должен учитывать ширину каждой ветви.

Первый способ требует лишь одного прохода по дереву. Однако для этого придется включить в результирующий SVG-файл посторонние атрибуты, необходимые только для служебных целей. Мы поместим эти атрибуты в отдельное пространство имен, чтобы исключить конфликты с атрибутами SVG:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  <!-- версия 1.1 объявлена не действующей, но Saxon поддерживает ее -->
  <!-- ради xsl:script. -->
  version="1.1"
  xmlns:emath="http://www.exslt.org/math"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:tree="http://www.ora.com/XSLTCookbook/ns/tree"
  xmlns:Math="java:java.lang.Math"
  extension-element-prefixes="Math"
  exclude-result-prefixes="Math emath">

  <xsl:script implements-prefix="Math"
                xmlns:Math="java:java.lang.Math"
                language="java"
                src="java:java.lang.Math"/>

  <xsl:include href="../math/math.max.xslt"/>

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"
    doctype-public="-//W3C//DTD SVG 1.0/EN"
    doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/
svg10.dtd"/>

  <!-- Эти параметры управляют линейными характеристиками дерева и его
узлов -->
  <xsl:variable name="width" select="500"/>
  <xsl:variable name="height" select="500"/>
  <xsl:variable name="nodeWidth" select="2"/>
  <xsl:variable name="nodeHeight" select="1"/>
  <xsl:variable name="horzSpace" select="0.5"/>
  <xsl:variable name="vertSpace" select="1"/>

  <xsl:template match="/">
```

[1] На самом деле делить надо на число узлов + 1, иначе получится так называемая «ошибка на 1».

```
<svg width="{$width}" height="{$height}">

  <!-- Запоминаем поддерево этого узла в переменной -->
  <xsl:variable name="subTree">
    <xsl:apply-templates/>
  </xsl:variable>

  <!-- maxPos — это максимум из двух самых отдаленных координат X и Y -->
 <!-- при рисовании узла -->
  <xsl:variable name="maxPos"
                select=" Math:max(number($subTree/g/@tree:MAXX),
                         number($subTree/g/@tree:MAXY))"/>
  <xsl:variable name="maxDim" select="Math:max($width,$height)"/>

  <!-- Масштабируем дерево, так чтобы все узлы оказались в области
 рисования -->
  <g transform="scale({$maxDim div ($maxPos + 1)})">

  <!-- Пользуйтесь функцией exsl:node-set($subTree) -->
  <!-- если ваш процессор XSLT поддерживает только версию 1.0 -->
    <xsl:copy-of select="$subTree/g/*"/>
  </g>

</svg>
</xsl:template>

<!-- Шаблон сопоставляется со всеми нелистовыми узлами -->
<xsl:template match="*[*]">

  <xsl:variable name="subTree">
      <xsl:apply-templates/>
  </xsl:variable>

  <!-- По горизонтали узел располагается так, чтобы он был центрирован -->
 <!-- над своими потомками -->
  <xsl:variable name="thisX"
                 select="sum($subTree/*/@tree:THISX)
                              div count($subTree/*)"/>


  <xsl:variable name="maxX" select="$subTree/*[last(  )]/@tree:MAXX"/>

  <!-- По вертикали узел располагается на своем уровне -->
  <xsl:variable name="thisY"
       select="($vertSpace + $nodeHeight) * count(ancestor-or-self::*)"/>

    <xsl:variable name="maxY">
```

```
        <xsl:call-template name="emath:max">
            <!-- Пользуйтесь функцией exsl:node-set($subTree) -->
        <!-- если ваш процессор XSLT поддерживает только версию 1.0 -->
        <xsl:with-param name="nodes" select="$subTree/*/@tree:MAXY"/>
      </xsl:call-template>
    </xsl:variable>

    <!-- Помещаем родителя, его детей и соединители в группу -->
    <!-- Включаем в эту группу также служебные атрибуты для передачи -->
    <!-- информации вверх по дереву -->
    <g tree:THISX="{$thisX}" tree:MAXX="{$maxX}" tree:MAXY="{$maxY}">
        <rect x="{$thisX - $nodeWidth}"
              y="{$thisY - $nodeHeight}"
              width="{$nodeWidth}"
              height="{$nodeHeight}"
               style="fill: none; stroke: black; stroke-width:0.1"/>

    <!-- Рисуем соединительную линию между текущим узлом и его
    потомками -->
    <xsl:call-template name="drawConnections">
         <xsl:with-param name="xParent" select="$thisX - $nodeWidth"/>
         <xsl:with-param name="yParent" select="$thisY - $nodeHeight"/>
         <xsl:with-param name="widthParent" select="$nodeWidth"/>
         <xsl:with-param name="heightParent" select="$nodeHeight"/>
         <xsl:with-param name="children" select="$subTree/g/rect"/>
    </xsl:call-template>

    <!-- Копируем SVG поддерева -->
    <xsl:copy-of select="$subTree"/>
  </g>

</xsl:template>

<!-- Этот шаблон сопоставляется с листовыми узлами -->
<xsl:template match="*">

  <!-- По горизонтали листовые узлы располагаются с учетом количества -->
  <!-- предшествующих листовых узлов -->
  <xsl:variable name="maxX"
       select="($horzSpace + $nodeWidth) *
                (count(preceding::*[not(child::*)] ) + 1) "/>
```

Можно было бы с помощью выражения `count(ancestor-or-self::*)` каждый раз вычислять уровень. Но проще и эффективнее передавать уровень в качестве дополнительного параметра:

```
  <!-- По вертикали узел располагается на своем уровне -->
  <xsl:variable name="maxY"
    select="($vertSpace + $nodeHeight) * count(ancestor-or-self::*) "/>

  <g tree:THISX="{$maxX}" tree:MAXX="{$maxX}" tree:MAXY="{$maxY}">
    <rect x="{$maxX - $nodeWidth}"
          y="{$maxY - $nodeHeight}"
          width="{$nodeWidth}"
          height="{$nodeHeight}"
          style="fill: none; stroke: black; stroke-width:0.1;"/>
  </g>

</xsl:template>

<!-- Переопределить в импортирующей таблице стилей, если желательно -->
<!-- изменить стиль рисования соединительных линий -->
<xsl:template name="drawConnections">
  <xsl:param name="xParent"/>
  <xsl:param name="yParent"/>
  <xsl:param name="widthParent"/>
  <xsl:param name="heightParent"/>
  <xsl:param name="children"/>
  <xsl:call-template name="drawSquareConnections">
    <xsl:with-param name="xParent" select="$xParent"/>
    <xsl:with-param name="yParent" select="$yParent"/>
    <xsl:with-param name="widthParent" select="$widthParent"/>
    <xsl:with-param name="heightParent" select="$heightParent"/>
    <xsl:with-param name="children" select="$children"/>
  </xsl:call-template>
</xsl:template>

<!-- Прямая соединительная линия — это кратчайший путь от середины -->
<!-- нижней стороны родителя до середины верхней стороны потомка -->
<xsl:template name="drawStraightConnections">
  <xsl:param name="xParent"/>
  <xsl:param name="yParent"/>
  <xsl:param name="widthParent"/>
  <xsl:param name="heightParent"/>
  <xsl:param name="children"/>
  <xsl:for-each select="$children">
     <line x1="{$xParent + $widthParent div 2}"
           y1="{$yParent + $heightParent}"
           x2="{@x + $nodeWidth div 2}"
           y2="{@y}"
```

```
            style="stroke: black; stroke-width:0.1;"/>
  </xsl:for-each>
</xsl:template>

<!-- Прямоугольная соединительная линия — это состоящий только из -->
<!-- горизонтальных и вертикальных отрезков кратчайший путь от -->
<!-- середины нижней стороны родителя до середины верхней стороны -->
<!-- потомка -->
<xsl:template name="drawSquareConnections">
  <xsl:param name="xParent"/>
  <xsl:param name="yParent"/>
  <xsl:param name="widthParent"/>
  <xsl:param name="heightParent"/>
  <xsl:param name="children"/>

  <xsl:variable name="midY"
      select="($children[1]/@y + ($yParent + $heightParent)) div 2"/>

  <!-- вертикальный отрезок, отходящий от родителя -->
  <line x1="{$xParent + $widthParent div 2}"
        y1="{$yParent + $heightParent}"
        x2="{$xParent + $widthParent div 2}"
        y2="{$midY}"
        style="stroke: black; stroke-width:0.1;"/>

  <!-- средний горизонтальный отрезок -->
  <line x1="{$children[1]/@x + $children[1]/@width div 2}"
        y1="{$midY}"
        x2="{$children[last()]/@x + $children[1]/@width div 2}"
        y2="{$midY}"
        style="stroke: black; stroke-width:0.1;"/>

  <!-- вертикальные отрезки, идущие к потомкам -->
  <xsl:for-each select="$children">
     <line x1="{@x + $nodeWidth div 2}"
           y1="{$midY}"
           x2="{@x + $nodeWidth div 2}"
           y2="{@y}"
           style="stroke: black; stroke-width:0.1;"/>
  </xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей рисует структуру произвольного XML-документа в виде дерева. На рис. 11.15 показан результат ее применения в простому входному XML-файлу.

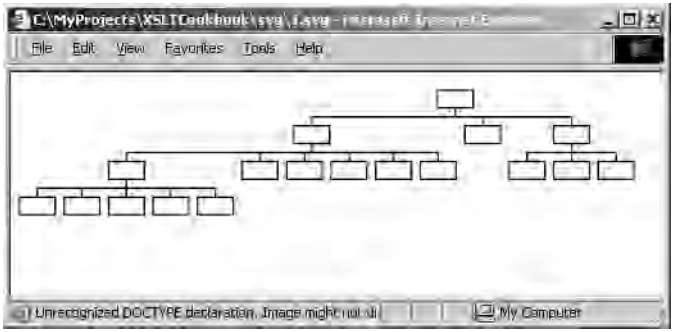


Рис. 11.15. Структура XML-документа, преобразованная в SVG

Первый алгоритм располагает родительские узлы посередине относительно множества их дочерних узлов. В результате для несбалансированных деревьев корневой узел оказывается не в центре. Следующий алгоритм несколько лучше, поскольку решает проблему смещения по горизонтали и не засоряет SVG посторонними атрибутами. Однако для него требуется два прохода по дереву:

```
<xsl:stylesheet version="1.1"
                xmlns:emath="http://www.exslt.org/math"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:tree="http://www.ora.com/XSLTCookbook/ns/tree"
                xmlns:Math="java:java.lang.Math"
                extension-element-prefixes="Math"
                exclude-result-prefixes="Math emath">

  <xsl:script implements-prefix="Math"
            xmlns:Math="java:java.lang.Math"
               language="java"
               src="java:java.lang.Math"/>

  <xsl:include href="../math/math.max.xslt"/>

  <xsl:output method="xml" version="1.0"
                encoding="UTF-8"
                indent="yes"
                doctype-public="-//W3C//DTD SVG 1.0/EN"
                doctype-system="http://www.w3.org/TR/2001/REC-SVG-
20010904/DTD/svg10.dtd"/>

  <xsl:variable name="width" select="500"/>
```

```
<xsl:variable name="height" select="500"/>
<xsl:variable name="nodeWidth" select="2"/>
<xsl:variable name="nodeHeight" select="1"/>
<xsl:variable name="horzSpace" select="0.5"/>
<xsl:variable name="vertSpace" select="1"/>
<xsl:variable name="strokeWidth" select="0.1"/>

<xsl:template match="/">

 <!-- На первом проходе входной документ копируется с добавлением -->
<!-- служебных атрибутов -->
  <xsl:variable name="treeWithLayout">
    <xsl:apply-templates mode="layout"/>
  </xsl:variable>

  <xsl:variable name="maxPos"
    select="Math:max($treeWithLayout/*/@tree:WEIGHT *
              ($nodeWidth + $horzSpace),
               $treeWithLayout/*/@tree:MAXDEPTH *
                 ($nodeHeight + $vertSpace))"/>

  <xsl:variable name="maxDim" select="Math:max($width,$height)"/>

  <xsl:variable name="scale" select="$maxDim div ($maxPos + 1)"/>

  <!-- На втором проходе создается SVG -->
  <svg height="{$height}" width="{$width}">
    <g transform="scale({$scale})">
      <xsl:apply-templates select="$treeWithLayout/*" mode="draw">
        <xsl:with-param name="x" select="0"/>
        <xsl:with-param name="y" select="0"/>
        <xsl:with-param name="width" select="$width div $scale"/>
        <xsl:with-param name="height" select="$height div $scale"/>
      </xsl:apply-templates>
    </g>
  </svg>
</xsl:template>

<!-- Размещаем узлы вместе с их потомками -->
<xsl:template match="node(  )[*]" mode="layout">
  <xsl:variable name="subTree">
    <xsl:apply-templates mode="layout"/>
  </xsl:variable>

  <!-- Нелистовым узлам присваивается вес, равный сумме весов их детей -->
```

```
    <xsl:variable name="thisWeight"
                        select="sum($subTree/*/@tree:WEIGHT)"/>

    <xsl:variable name="maxDepth">
      <xsl:call-template name="emath:max">
        <xsl:with-param name="nodes"
                                select="$subTree/*/@tree:MAXDEPTH"/>
      </xsl:call-template>
    </xsl:variable>

    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <xsl:attribute name="tree:WEIGHT">
        <xsl:value-of select="$thisWeight"/>
      </xsl:attribute>
      <xsl:attribute name="tree:MAXDEPTH">
        <xsl:value-of select="$maxDepth"/>
      </xsl:attribute>
      <xsl:copy-of select="$subTree"/>
    </xsl:copy>

  </xsl:template>

  <!-- Размещаем листовые узлы -->
  <xsl:template match="*" mode="layout">
    <xsl:variable name="depth" select="count(ancestor-or-self::*) "/>
    <xsl:copy>
      <xsl:copy-of select="@*"/>
      <!-- Листовым узлам присваивается вес 1 -->
      <xsl:attribute name="tree:WEIGHT">
        <xsl:value-of select="1"/>
      </xsl:attribute>
      <xsl:attribute name="tree:MAXDEPTH">
        <xsl:value-of select="$depth"/>
      </xsl:attribute>
    </xsl:copy>
  </xsl:template>

  <!-- Рисуем нелистовые узлы -->
  <xsl:template match="node(  )[*]" mode="draw">
    <xsl:param name="x"/>
    <xsl:param name="y"/>
    <xsl:param name="width"/>
    <xsl:variable name="thisX"
          select="$x + $width div 2 - ($nodeWidth+$horzSpace) div 2"/>
```

```
    <xsl:variable name="subTree">
      <xsl:call-template name="drawSubtree">
        <xsl:with-param name="nodes" select="*"/>
        <xsl:with-param name="weight" select="@tree:WEIGHT"/>
        <xsl:with-param name="x" select="$x"/>
        <xsl:with-param name="y" select="$y + $nodeHeight + $vertSpace"/>
        <xsl:with-param name="width" select="$width"/>
      </xsl:call-template>
    </xsl:variable>
    <g>

      <rect x="{$thisX}"
               y="{$y}"
                width="{$nodeWidth}"
                height="{$nodeHeight}"
                style="fill: none; stroke: black; stroke-
width:{$strokeWidth};"/>

      <xsl:call-template name="drawConnections">
        <xsl:with-param name="xParent" select="$thisX"/>
        <xsl:with-param name="yParent" select="$y"/>
        <xsl:with-param name="widthParent" select="$nodeWidth"/>
        <xsl:with-param name="heightParent" select="$nodeHeight"/>
        <xsl:with-param name="children" select="$subTree/g/rect"/>
      </xsl:call-template>

      <xsl:copy-of select="$subTree"/>

    </g>

  </xsl:template>

  <!-- Рисуем листовые узлы -->
  <xsl:template match="*" mode="draw">
    <xsl:param name="x"/>
    <xsl:param name="y"/>
    <xsl:param name="width"/>
    <xsl:variable name="thisX"
            select="$x + $width div 2 - ($nodeWidth+$horzSpace) div 2"/>
    <g>
      <rect x="{$thisX}"
         y="{$y}"
         width="{$nodeWidth}"
         height="{$nodeHeight}"
         style="fill: none; stroke: black; stroke-width:{$strokeWidth};"/>
```

```
    </g>
  </xsl:template>

  <!-- Рекурсивная процедура для рисования поддерева -->
  <!-- Распределяет горизонтальное пространство с учетом веса узла -->
  <xsl:template name="drawSubtree">
    <xsl:param name="nodes" select="/.."/>
    <xsl:param name="weight"/>
    <xsl:param name="x"/>
    <xsl:param name="y"/>
    <xsl:param name="width"/>

    <xsl:if test="$nodes">
      <xsl:variable name="node" select="$nodes[1]"/>
      <xsl:variable name="ratio" select="$node/@tree:WEIGHT div $weight"/>

      <!-- Рисуем узел и его детей в выделенной области, -->
      <!-- основываясь на текущем значении x и вычисленной ширине -->
      <!-- области -->
      <xsl:apply-templates select="$node" mode="draw">
        <xsl:with-param name="x" select="$x"/>
        <xsl:with-param name="y" select="$y"/>
        <xsl:with-param name="width" select="$width * $ratio"/>
      </xsl:apply-templates>

      <!-- Обрабатываем остальные узлы -->
      <xsl:call-template name="drawSubtree">
        <xsl:with-param name="nodes" select="$nodes[position(  ) > 1]"/>
        <xsl:with-param name="weight" select="$weight"/>
        <xsl:with-param name="x" select="$x + $width * $ratio"/>
        <xsl:with-param name="y" select="$y"/>
        <xsl:with-param name="width" select="$width"/>
      </xsl:call-template>
    </xsl:if>

  </xsl:template>

<!-- Код рисования соединительных линий опущен, т.к. не отличается -->
<!-- от предыдущего примера. -->

</xsl:stylesheet>
```

На рис. 11.16 показано, как тот же самый входной документ рисуется новым алгоритмом.

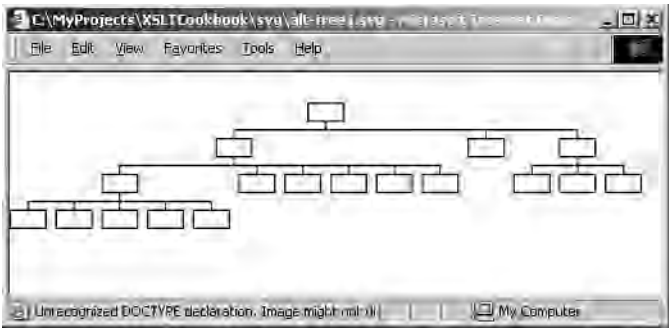


Рис. 11.16. Более сбалансированное представление структуры XML-документа

Обсуждение

Предыдущие рецепты не полны, так как позволяют нарисовать только скелет дерева без содержимого. Очевидное улучшение – добавить к узлам дерева текст, чтобы можно было распознать, к чему они относятся. Это не просто, поскольку SVG не масштабирует текст автоматически. Дополнительную сложность вносит тот факт, что ширина прямоугольников становится зависимой от объема размещенного в них текста. См. рецепт 14.14 и написанную на Java функцию расширения, которая помогает решить проблему размещения текста в SVG.

Мы решили отобразить все узлы входного документа на узлы SVG-дерева. На практике, вероятно, следовало бы отфильтровать ненужные узлы, указав вместо выражений `match="node( )[*]"` и `match="*"` другие образцы.

Если структура дерева не должна буквально повторять иерархическую структуру XML-документа, то следует выполнить предварительную обработку с целью преобразования структуры.

В таблицы стилей включен код для поддержки двух стилей соединительных линий. В примерах мы пользовались прямоугольными соединениями. Чтобы нарисовать прямые соединения, как показано на рис. 11.17, нужно переопределить `drawConnections`, так чтобы вызывался шаблон `drawStraightConnections`.

С точки зрения переносимости в этих таблицах стилей есть две проблемы. Во-первых, в них вызывается написанная на Java функция расширения `Math:max`. Ее легко можно реализовать и на XSLT. Однако в таблицах стилей для генерации SVG часто бывают нужны и другие расширения, так что в общем случае их использование неизбежно. Вторая проблема – предположение о поддержке спецификации XSLT 1.1 (ныне объявленной недействующей) или более поздней, чтобы фрагменты результирующего дерева можно было корректно рассматривать как наборы узлов. Можно вместо этого воспользоваться теми средствами конвертации наборов узлов, которые имеются в доступном вам процессоре XSLT.

См. также

Дополнительную информацию о сложных алгоритмах рисования деревьев и графов более общего вида можно почерпнуть в книге Giuseppe Di Battista,

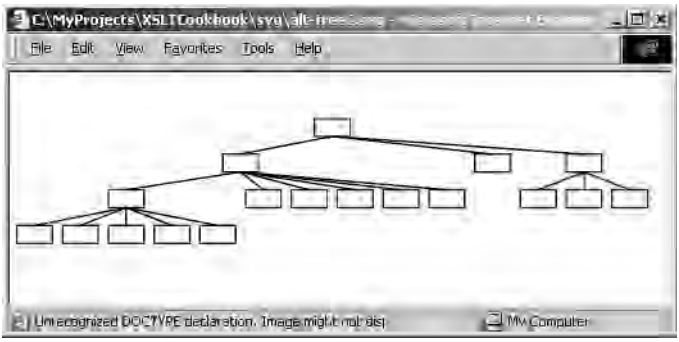


Рис. 11.17. Структура XML-документа, преобразованная в SVG с использованием прямых соединений

Peter Eades, Roberto Tamassia, and Ionnis G. Tollis *Graph Drawing: Algorithms for the Visualization of Graphs* (Prentice Hall, 1999). Но имейте в виду, что она насыщена нетривиальной математикой; это не просто сборник алгоритмов на псевдокоде.

11.4. Создание интерактивных Web-страниц, включающих SVG

Задача

Требуется встроить в HTML-код куски на языке SVG, чтобы пользователь мог взаимодействовать со страницей.

Решение

В основе этого решения лежит код из статьи Дидье Мартина (Didier Martin) на сайте XML.com *Integration by Parts: XSLT, XLink and SVG* (*http://www.xml.com/lpt/a/2000/03/ 22/style/index.html*). Таблица стилей встраивает в HTML-страницу SVG-графику вкупе с информацией, полученной из XML-документа. Для взаимодействия с графикой включен также сценарий на языке JavaScript. Данный пример может служить прототипом для сайтов по торговле недвижимостью, позволяющих пользователю взаимодействовать с чертежом дома.

Входной XML-файл содержит информацию о доме. С каждой комнатой ассоциирован атрибут `id`, который связывает данные об этой комнате с элементом `g` на SVG-диаграмме с тем же идентификатором.

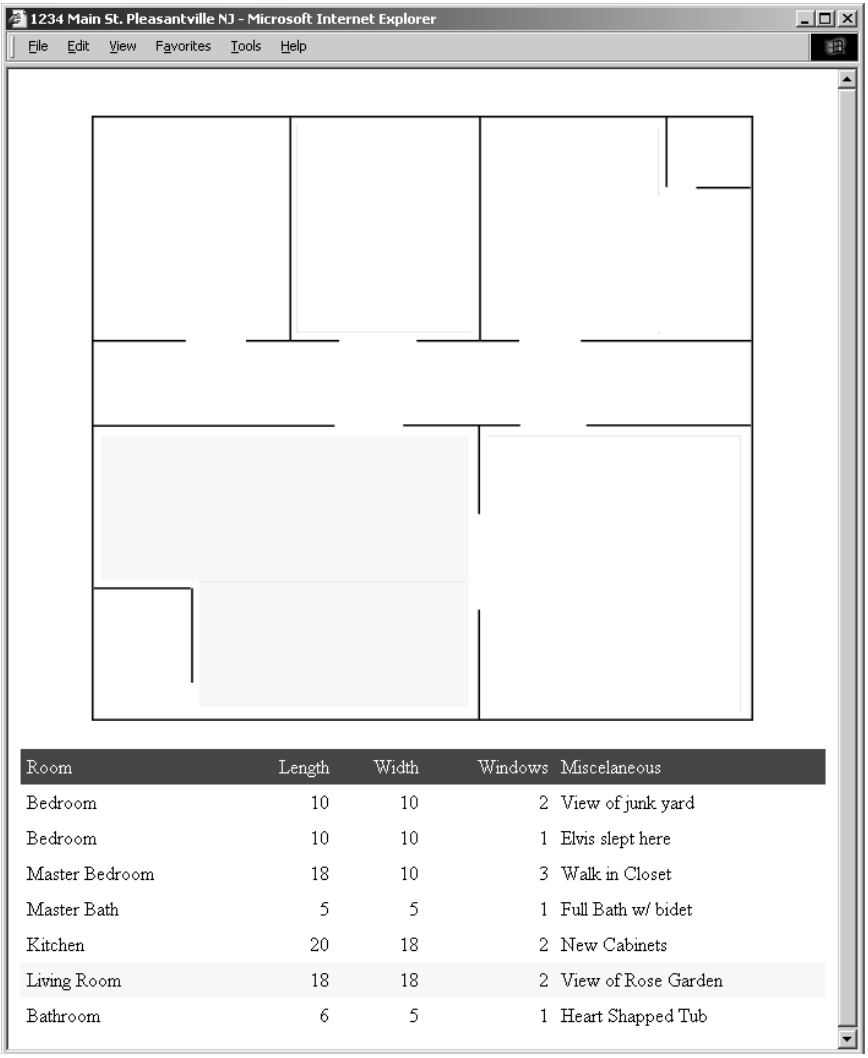
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="HouseLayout.xsl"?>
<House>
  <Location>
      <Address>1234 Main St. </Address>
      <City>Pleasantville </City>
```

```
    <State>NJ</State>
</Location>
<Layout figure="HouseLayout.svg">
    <Room id="bedroom1">
        <Name>Спальня</Name>
        <Length>10</Length>
        <Width>10</Width>
        <Windows>2</Windows>
        <Misc>Вид на помойку</Misc>
    </Room>
    <Room id="bedroom2">
        <Name>Спальня</Name>
        <Length>10</Length>
        <Width>10</Width>
        <Windows>1</Windows>
        <Misc>Здесь спал Элвис</Misc>
    </Room>
    <Room id="masterBedroom">
        <Name>Хозяйская спальня</Name>
        <Length>18</Length>
        <Width>10</Width>
        <Windows>3</Windows>
        <Misc>Проход в кабинет</Misc>
    </Room>
    <Room id="masterBath">
        <Name>Хозяйская ванная</Name>
        <Length>5</Length>
        <Width>5</Width>
        <Windows>1</Windows>
        <Misc>Полностью оборудованная ванная, включая биде</Misc>
    </Room>
    <Room id="kitchen">
        <Name>Кухня</Name>
        <Length>20</Length>
        <Width>18</Width>
        <Windows>2</Windows>
        <Misc>Новые шкафчики</Misc>
    </Room>
    <Room id="livingRoom">
        <Name>Гостиная</Name>
        <Length>18</Length>
        <Width>18</Width>
        <Windows>2</Windows>
        <Misc>Вид на розовый сад</Misc>
```

```
        </Room>
        <Room id="bath1">
            <Name>Ванная</Name>
            <Length>6</Length>
            <Width>5</Width>
            <Windows>1</Windows>
            <Misc>Ванна в форме сердца</Misc>
        </Room>
    </Layout>
</House>
```

Показанная ниже таблица стилей включает SVG-файл, преобразует XML-данные в табличную форму и добавляет JavaScript-сценарий, делающий страницу интерактивной (рис. 11.18).

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          version="1.0">
<xsl:output method="html" version="4"/>

<xsl:template match="/">
<html>
<head>
<title><xsl:value-of select="concat(*/*/Address,*/*/City,*/*/State)"/></
title>
<script><![CDATA[

var item_selected = null;

// Эта функция вызывается в ответ на событие mouseover.
// Для доступа к узлам дерева документа мы используем
// SVGDOM и XML DOM. Конкретно, функция изменяет
// элементы, идентифицируемые атрибутом id.
// Отметим, что для изменения атрибута style в SVG DOM
// не нужно заранее знать значение этого атрибута.
// Напротив, в случае XML DOM необходимо знать все
// содержимое атрибута style.
function on_mouse_over (ID)
{
  if (ID == item_selected)
          return true;

  var obj_name = ID ;

  // Изменить стиль SVG-элемента
  // -----------------
```



Room	Length	Width	Windows	Miscelaneous
Bedroom	10	10	2	View of junk yard
Bedroom	10	10	1	Elvis slept here
Master Bedroom	18	10	3	Walk in Closet
Master Bath	5	5	1	Full Bath w/ bidet
Kitchen	20	18	2	New Cabinets
Living Room	18	18	2	View of Rose Garden
Bathroom	6	5	1	Heart Shapped Tub

Рис. 11.18. Интерактивный SVG-рисунок, сгенерированный из XML-документа

```
// 1 - получить документый элемент SVGDOM от программы просмотра
// Adobe SVG viewer.
// 2 - получить элемент SVG-документа по идентификатору.
// 3 - получить атрибут style найденного элемента SVG DOM.
// Функция getStyle ориентирована исключительно на SVG DOM.
// Функция getStyle возвращает объект, описывающий стиль.
// В возвращенном объекте мы изменяем атрибут стиля 'fill'.
// Отметим, что в отличие от XML DOM не нужно заранее знать
// содержимое атрибута style, чтобы изменить одно из CSS-свойств.
```

```
  var svgdoc = document.figure.getSVGDocument(  );
  var svgobj = svgdoc.getElementById(obj_name);
  if (svgobj != null)
  {
      var svgStyle = svgobj.getStyle(  );
      svgStyle.setProperty ('fill', 'yellow');
  }

  // Вот что должно было бы быть, если бы браузер полностью
  // поддерживал XML DOM.
  // --------------------------
  // Получить элемент HTML-документа (из раздела body) с известным
  // идентификатором. Изменить атрибут style этого элемента с
  // помощью XML DOM. Отметим, что в отличие от SVG DOM атрибут
  // style изменяется целиком, а не только значение одного из
  // записанных в нем CSS-свойств.
  // НЕ РАБОТАЕТ...
  var svgdesc = document.getElementById(obj_name);
  if (svgdesc != null)
      svgdesc.setAttribute("style", "background-color:yellow; cursor:hand");

  // Вот что мы делаем для IE 5 DHTML DOM
  // ----------------------
  var DHTMLobj = document.all.item(obj_name)
  if (DHTMLobj != null)
         DHTMLobj.style.backgroundColor = "yellow";
  return true;
}

// Эта функция вызывается в ответ на событие mouseout.
// Для доступа к узлам дерева документа мы используем
// SVGDOM и XML DOM. Конкретно, функция изменяет
// элементы, идентифицируемые атрибутом id.
// Отметим, что для изменения атрибута style в SVG DOM
// не нужно заранее знать значение этого атрибута.
// Напротив, в случае XML DOM необходимо знать все
function on_mouse_out (ID)
{
  if (ID =  = item_selected)
         return true;

  var obj_name = ID ;

  // Изменить стиль элемента SVG
  // -----------------
```

```
  // 1 - получить документный элемент SVGDOM от программы просмотра
  // Adobe SVG viewer.
  // 2 - получить элемент SVG-документа по идентификатору.
  // 3 - получить атрибут style найденного элемента SVG DOM.
  // Функция getStyle ориентирована исключительно на SVG DOM.
  // Функция getStyle возвращает объект, описывающий стиль.
  // В возвращенном объекте мы изменяем атрибут стиля 'fill'.
  // Отметим, что в отличие от XML DOM не нужно заранее знать
  // содержимое атрибута style, чтобы изменить одно из CSS-свойств.
  var svgdoc = document.figure.getSVGDocument(  );
  var svgobj = svgdoc.getElementById(obj_name);
  if (svgobj != null)
  {
         var svgStyle = svgobj.getStyle(  );
         svgStyle.setProperty ('fill', 'white');
         svgStyle.setProperty ('stroke', 'white');
  }

  // Вот что должно было бы быть, если бы браузер полностью
  // поддерживал XML DOM.
  // --------------------------
  // Получить элемент HTML-документа (из раздела body) с известным
  // идентификатором. Изменить атрибут style этого элемента с
  // помощью XML DOM. Отметим, что в отличие от SVG DOM атрибут
  //  style изменяется целиком, а не только значение одного из
  // записанных в нем CSS-свойств.
  // НЕ РАБОТАЕТ...
  var svgdesc = document.getElementById(obj_name);
  if (svgdesc != null)
      svgdesc.setAttribute("style", "background-color:white;");

  // Вот что мы делаем для IE 5 DHTML DOM
  // --------------------
  var DHTMLobj = document.all.item(obj_name)
  if (DHTMLobj != null)
         DHTMLobj.style.backgroundColor = "white";

  return true;
}
function on_mouse_click(ID)
{
     var obj_name = ID ;

     // восстановить цвет ранее выбранной комнаты
     if (item_selected)
```

```
        {
            var svgdoc = document.figure.getSVGDocument(  );
            var svgobj = svgdoc.getElementById(obj_name);
            if (svgobj != null)
            {
                  var svgStyle = svgobj.getStyle(  );
                  svgStyle.setProperty ('fill', 'white');
            }
            var DHTMLobj = document.all.item(obj_name)
            if (DHTMLobj != null)
            {
                  DHTMLobj.style.backgroundColor = "white";
                  DHTMLobj.style.fontWeight  = "normal";
            }
        }
        // Выбрать новую комнату
        if (item_selected != ID)
        {
            var svgdoc = document.figure.getSVGDocument(  );
            var svgobj = svgdoc.getElementById(obj_name);
            if (svgobj != null)
            {
                  var svgStyle = svgobj.getStyle(  );
                  svgStyle.setProperty ('fill', '#C0C0C0');
            }
            var DHTMLobj = document.all.item(obj_name)
            if (DHTMLobj != null)
            {
                  DHTMLobj.style.backgroundColor = "#C0C0C0";
                  DHTMLobj.style.fontWeight  = "bolder";
            }
            item_selected = ID;
        }
        else
            item_selected = null;

        return true;
}
]]></script>
</head>

<body>
      <xsl:apply-templates/>
</body>
```

```
</html>
</xsl:template>

<xsl:template match="Layout">
     <div align="center">
          <embed name="figure" width="540" height="540" type="image/svg"
          pluginspage="http://www.adobe.com/svg/viewer/install/">
          <xsl:attribute name="src"><xsl:value-of select="@figure"/>
         </xsl:attribute>
          </embed>
     </div>
     <table border="0" cellpadding="1" cellspacing="0" width="100%"
bgcolor="black">
     <tr>
          <table border="0" cellpadding="5" cellspacing="0" width="100%"
          bgcolor="white">
               <tr style="background-color:#990033; color:white;">
                   <td>Room</td>
                    <td align="right">Length</td>
                    <td align="right">Width</td>
                    <td align="right">Windows</td>
                    <td>Miscelaneous</td>
               </tr>
               <xsl:apply-templates/>
          </table>
     </tr>
</table>
</xsl:template>

<xsl:template match="Room">
    <tr id="{@id}" style="'background-color:white;'"
        onmouseover="on_mouse_over('{@id}')"
        onmouseout="on_mouse_out('{@id}')"
        onclick="on_mouse_click('{@id}')">
     <td><xsl:value-of select="Name"/></td>
     <td align="right"><xsl:value-of select="Length"/></td>
     <td align="right"><xsl:value-of select="Width"/></td>
     <td align="right"><xsl:value-of select="Windows"/></td>
     <td><xsl:value-of select="Misc"/></td>
    </tr>
</xsl:template>

<xsl:template match="text(  )"/>

</xsl:stylesheet>
```

Обсуждение

В предыдущих примерах мы занимались получением SVG из XML, а в этом интегрировали SVG в приложение, включающее и другие Web-технологии. Мы лишь слегка коснулись того, что можно достичь в таких приложениях. В SVG есть средства для анимации и динамического создания контента. В сочетании с XSLT-преобразованиями результаты могут оказаться очень впечатляющими. Взгляните на следующую таблицу стилей, которая основана на графических примитивах из рецепта 11.2, но еще и позволяет пользователю взаимодействовать с графиком:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svgu="http://www.ora.com/XSLTCookbook/ns/svg-utils"
  xmlns:test="http://www.ora.com/XSLTCookbook/ns/test"
  exclude-result-prefixes="svgu test">

<xsl:import href="svg-utils.xslt"/>

<xsl:output method="html"/>

<test:data>1.0</test:data>
<test:data>2.0</test:data>
<test:data>3.0</test:data>
<test:data>4.0</test:data>
<test:data>5.0</test:data>
<test:data>13.0</test:data>
<test:data>2.7</test:data>
<test:data>13.9</test:data>
<test:data>22.0</test:data>
<test:data>8.5</test:data>

<xsl:template match="/">
<html>
  <head>
      <title>Интерактивная столбчатая диаграмма</title>
      <object id="AdobeSVG"
          classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"/>
      <xsl:processing-instruction name="import">
          <xsl:text>namespace="svg" implementation="#AdobeSVG"</xsl:text>
      </xsl:processing-instruction>
<script>
<![CDATA[
```

```
function on_change (ID,VALUE)
{
  // Получить SVG-документ
  var svgDocument = document.all.item('figure').getSVGDocument(  );

  // Идентификаторы столбцов устроены так: префикс контекста + _bar_ + ID
  var barName = "interact_bar_" + ID ;

  var barObj = svgDocument.getElementById(barName);
  if (barObj != null)
  {
      barObj.setAttribute('y2', VALUE);
  }

  return true;
}

]]></script>
</head>
<body>
  <div align="center">
     <svg:svg width="400" height="400" id="figure">
      <xsl:call-template name="svgu:bars">
          <xsl:with-param name="data" select="document('')/*/test:data"/>
          <xsl:with-param name="width" select=" '300' "/>
          <xsl:with-param name="height" select=" '350' "/>
          <xsl:with-param name="offsetX" select=" '50' "/>
          <xsl:with-param name="offsetY" select=" '25' "/>
          <xsl:with-param name="boundingBox" select="1"/>
          <xsl:with-param name="max" select="25"/>
          <xsl:with-param name="context" select=" 'interact' "/>
          </xsl:call-template>
      </svg:svg>
  </div>
  <table border="1" cellspacing="1" cellpadding="1">
        <tbody>
         <xsl:for-each select="document('')/*/test:data">
           <xsl:variable name="pos" select="position(  )"/>
           <xsl:variable name="last" select="last(  )"/>
           <tr>
               <td>Bar <xsl:value-of select="$pos"/></td>
               <td>
                   <input type="text">
                         <xsl:attribute name="value">
```

```
                <xsl:value-of select="."/>
              </xsl:attribute>
              <xsl:attribute name="onchange">
                <xsl:text>on_change(</xsl:text>
                <!-- Вертикально ориентированные столбцы -->
                <!-- поворачиваются, поэтому идентификаторы нужно -->
                <!-- обратить. Пояснения см. в реализации svgu:bars. -->
                <xsl:value-of select="$last - $pos + 1"/>
                <xsl:text>, this.value)</xsl:text>
              </xsl:attribute>
            </input>
            </td>
          </tr>
        </xsl:for-each>
      </tbody>
    </table>
  </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей генерирует Web-страницу, которая позволяет изменять данные, и при этом изменяется высота столбцов. Здесь же демонстрируется техника внедрения SVG в HTML. К сожалению, работает этот пример только в браузерах IE 5.5 и более поздних и требует наличия надстройки Adobe SVG[1].

См. также

Статья Дидье Мартина на сайте XML.com *Integration by Parts: XSLT, XLink and SVG* (*http://www.xml.com/lpt/a/2000/03/22/style/index.html*) содержит более интересный пример взаимодействия с CAD-чертежом сложной детали.

В книге J. David Eisenberg *SVG Essentials* (O'Reilly, 2002) приводится подробная информация об анимации и написании сценариев в SVG.

Разработчикам, знакомым с языком Java и желающим серьезно заняться программированием для SVG, настоятельно рекомендуется заглянуть на сайт Apache Batik (*http://xml.apache.org/batik/*).

[1] Разумеется, у большинства современных компьютеров именно такая конфигурация. В будущих версиях Firefox (возможно, 1.1) поддержка SVG будет встроенной.

Глава 12. Генерация кода

> Хорошие программисты пишут хороший код. Великие программисты пишут программы для генерации кода.
>
> *Автор неизвестен*

12.0. Введение

Автоматизация – это Святой Грааль в разработке программного обеспечения. Вообще, прогресс в этой области в значительной мере обязан идее генерации кода по той или иной спецификации. Разве не этим занимаются ассемблеры и компиляторы? Но есть и другой вид генерации кода, когда целью является не исполняемый машинный код, а программа на языке высокого уровня, например, Java или C++. Зачем это может понадобиться и какое к этому отношение имеет XML?

При написании программ вы записываете различные знания на языке со специфическими синтаксическими правилами, оптимизированном для одного конкретного этапа жизненного цикла разработки. Проделанной работой трудно воспользоваться для выполнения других важных задач разработки, поскольку синтаксический разбор языков программирования сложен, а наиболее интересная информация упрятана в неформализованных комментариях. Представление знаний о приложении в формате XML открывает больше возможностей для ее использования в разных местах. Имея спецификацию, составленную на XML, вы можете сгенерировать код самого приложения, тестовые программы, документацию и, возможно, даже тестовые данные. Я не хочу сказать, что XML дает все это задаром. Как и всегда при разработке программного обеспечения, нужно сначала все тщательно спланировать, выстроить инфраструктуру, а потом уже пожинать плоды.

Эта глава отличается от остальных тем, что большинство примеров в ней – компоненты решения в контексте конкретного приложения. Для такого построения есть две причины.

Во-первых, маловероятно, что вы будете кодировать информацию в формате XML для последующей генерации кода только потому, что XML – это «круто». Как правило, предстоит решить некую крупную задачу, в которой XML можно использовать и для других целей. Поэтому и примеры в этом разделе станут более осмысленными, если представить их в контексте объемлющей задачи.

Во-вторых, эта объемлющая задача обычно возникает при разработке крупномасштабного приложения, что само по себе может заинтересовать читателя. Но даже если это не так, включение в рассмотрение объемлющей задачи не заведет нас слишком далеко от применения излагаемых идей к прочим этапам разработки.

Итак, в чем же состоит объемлющая задача?

Представим себе сложное клиент-серверное приложение. Говоря *сложное*, я имею в виду, что оно состоит из многочисленных процессов на стороне сервера и клиента. Эти процессы обмениваются между собой данными в виде сообщений, передаваемых с помощью связующего программного обеспечения для передачи сообщений (от точки к точке, типа публикация/подписка или то и другое). В качестве примеров таких программ можно назвать IBM MQSeries, Microsoft Message Queuing (MSMQ), BEA Systems Tuxedo и TIBCO Rendezvous. Для нас неважно, какой конкретно продукт используется. А важно то, что основную работу система выполняет при получении сообщения, в ответ на которое должна послать в ответ одно или несколько сообщений[1]. Сообщение может содержать XML (в виде SOAP), неразмеченный текст или двоичные данные. В главе 12 мы рассмотрим протокол SOAP в контексте WSDL. А сейчас нас интересуют межсерверные коммуникации, в которых XML применяется реже.

В таких сложных системах плохо то, что в них невозможно разобраться, просто изучив исходный текст какого-то одного компонента. Сначала нужно понять, как устроен обмен данными между процессами, или *межпроцессные протоколы сообщений*. Мы пойдем даже дальше и скажем, что в первом приближении детали отдельных процессов вообще несущественны. Можно рассматривать каждый процесс как черный ящик. И тогда, вместо того чтобы читать сотни тысяч строк кода, составляющего систему, можно начать ее изучение с анализа относительно небольшого набора сообщений, которыми обмениваются процессы.

Но тут же возникает следующий вопрос: как разобраться в языке межпроцессного общения в сложном приложении? Есть ли какое-то одно место, где можно почерпнуть всю информацию? Увы, зачастую это не так. Очень редко удается найти актуальную и полную спецификацию прикладных протоколов. Отыскать кусочки головоломки можно в различных заголовочных файлах и в проектной документации, разрабатывавшейся на протяжении жизненного цикла системы, но единого места, в котором была бы собрана вся жизненно важная информация, не существует. Во многих случаях единственный надежный способ получить ее – изучить исходный код, от чего я как раз и предостерегал в самом начале!

Ну хорошо, а какое отношение все это имеет к XML, XSLT и, главное, к генерации кода? Собственно, проблема состоит в отсутствии документации, детально описывающей структуру межпроцессных сообщений в приложении. Как мог бы

[1] Очевидно, что важны также входные данные от пользователей и формируемая для них выходная информация. Но ввод/вывод тоже можно считать частным случаем обмена сообщениями. Только такие сообщения обычно посылаются и принимаются не с помощью технологий межпроцессных коммуникаций, а по другим каналам.

выглядеть такой документ? Разработчики могли бы поддерживать в актуальном состоянии созданный в MS Word документ, в котором описаны все сообщения. Еще лучше, если бы этой теме был посвящен специальный сайт, поддерживающий навигацию и поиск. А, быть может (и вы, конечно, сами догадались!), информацию стоит хранить в XML-файле! Тогда из этого файла можно было бы сгенерировать и сайт. И уж раз такой файл имеется, то почему бы не попытаться сгенерировать хотя бы частично код, необходимый для обработки описанных сообщений. Именно этим мы и займемся в настоящей главе. Я буду называть множество XML-файлов *репозиторием межпроцессных сообщений*. В рецептах ниже демонстрируется, как генерировать код на основе такого репозитория.

Но, прежде чем переходить к рецептам, поговорим о структуре репозитория. Формально она описывается в виде схемы W3C XSD Schema, но мы представим только интуитивно понятный рисунок для тех, кто со схемами XML не знаком.

Рис. 12.1 был создан программой Altova's XML Spy 4.0 (*http://www.xmlspy.com*). Значок троеточия обозначает упорядоченную последовательность. Значок, напоминающий трехпозиционный переключатель, обозначает выбор.

Хотя этой схемы и достаточно для иллюстрации интересных рецептов генерации кода, до промышленного репозитория сообщений она не дотягивает. В репозитории можно было бы хранить следующие дополнительные данные:

- символические константы, описывающие размеры массивов и строк, а также значения, употребляемые в перечислениях;
- информацию о сложных типах данных, например, объединениях и псевдонимах типов (typedef в языке C);
- информацию о протоколах (последовательностях сообщений, которыми процессы обмениваются для достижения конкретного результата);
- информацию об авторах оригинальной версии, датах и авторах изменений и т.п.;
- информацию о доставке и транспортировке: имена издателей и подписчиков либо имена очередей.

В качестве примера рассмотрим простое клиент-серверное приложение, которое позволяет отправлять заказы на склад или отменять их. Репозиторий для подобного приложения мог бы выглядеть так:

```
<MessageRepository xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation="C:\MyProjects\XSLT Cookbook\code
gen\MessageRepository.xsd">
  <DataTypes>
    <Primitive>
      <Name>Real</Name>
      <Size>8</Size>
      <Category>real</Category>
    </Primitive>
    <Primitive>
      <Name>Integer</Name>
```

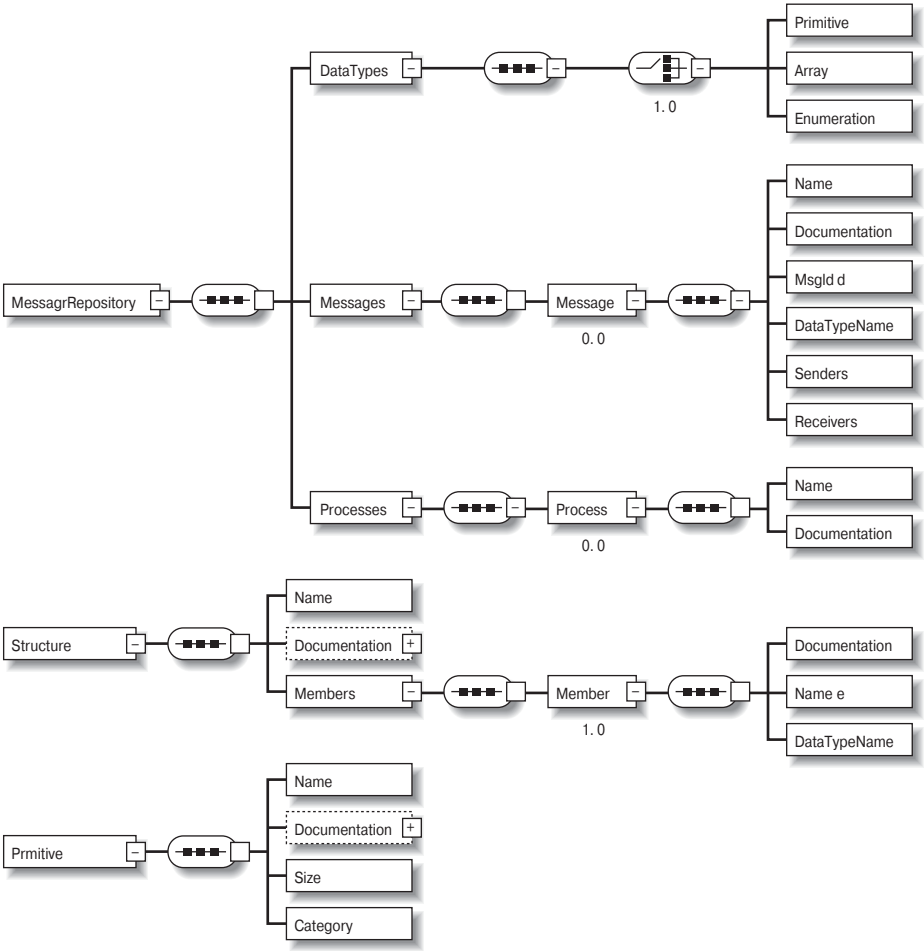


Рис. 12.1. Графическое представление XSD-схемы репозитория

```
  <Size>4</Size>
  <Category>signed integer</Category>
</Primitive>
<Primitive>
  <Name>StkSymbol</Name>
  <Size>10</Size>
  <Category>string</Category>
</Primitive>
<Primitive>
  <Name>Message</Name>
  <Size>100</Size>
  <Category>string</Category>
</Primitive>
```

```
<Primitive>
  <Name>Shares</Name>
  <Size>4</Size>
  <Category>signed integer</Category>
</Primitive>
<Enumeration>
  <Name>BuyOrSell</Name>
  <Enumerators>
    <Enumerator>
      <Name>BUY</Name>
      <Value>0</Value>
    </Enumerator>
    <Enumerator>
      <Name>SELL</Name>
      <Value>1</Value>
    </Enumerator>
  </Enumerators>
</Enumeration>
<Enumeration>
  <Name>OrderType</Name>
  <Enumerators>
    <Enumerator>
      <Name>MARKET</Name>
      <Value>0</Value>
    </Enumerator>
    <Enumerator>
      <Name>LIMIT</Name>
      <Value>1</Value>
    </Enumerator>
  </Enumerators>
</Enumeration>
<Structure>
  <Name>TestData</Name>
  <Members>
    <Member>
      <Name>order</Name>
      <DataTypeName>AddStockOrderData</DataTypeName>
    </Member>
    <Member>
      <Name>cancel</Name>
      <DataTypeName>CancelStockOrderData</DataTypeName>
    </Member>
  </Members>
</Structure>
```

```
<Structure>
  <Name>AddStockOrderData</Name>
  <Documentation>Запрос на добавление нового заказа.</Documentation>
  <Members>
    <Member>
      <Name>symbol</Name>
      <DataTypeName>StkSymbol</DataTypeName>
    </Member>
    <Member>
      <Name>quantity</Name>
      <DataTypeName>Shares</DataTypeName>
    </Member>
    <Member>
      <Name>side</Name>
      <DataTypeName>BuyOrSell</DataTypeName>
    </Member>
    <Member>
      <Name>type</Name>
      <DataTypeName>OrderType</DataTypeName>
    </Member>
    <Member>
      <Name>price</Name>
      <DataTypeName>Real</DataTypeName>
    </Member>
  </Members>
</Structure>
<Structure>
  <Name>AddStockOrderAckData</Name>
  <Documentation>Подтверждение успешного добавления заказа.
  </Documentation>
  <Members>
    <Member>
      <Name>orderId</Name>
      <DataTypeName>Integer</DataTypeName>
    </Member>
  </Members>
</Structure>
<Structure>
  <Name>AddStockOrderNackData</Name>
  <Documentation>Сообщение об ошибке при добавлении заказа.
  </Documentation>
  <Members>
    <Member>
      <Name>reason</Name>
```

```
          <DataTypeName>Message</DataTypeName>
        </Member>
      </Members>
    </Structure>
    <Structure>
      <Name>CancelStockOrderData</Name>
      <Documentation>Запрос на полную или частичную отмену заказа</
Documentation>
      <Members>
        <Member>
          <Name>orderId</Name>
          <DataTypeName>Integer</DataTypeName>
        </Member>
        <Member>
          <Name>quantity</Name>
          <DataTypeName>Shares</DataTypeName>
        </Member>
      </Members>
    </Structure>
    <Structure>
      <Name>CancelStockOrderAckData</Name>
      <Documentation>Подтверждение успешной отмены заказа.
      </Documentation>
      <Members>
        <Member>
          <Name>orderId</Name>
          <DataTypeName>Integer</DataTypeName>
        </Member>
        <Member>
          <Name>quantityRemaining</Name>
          <DataTypeName>Shares</DataTypeName>
        </Member>
      </Members>
    </Structure>
    <Structure>
      <Name>CancelStockOrderNackData</Name>
     <Documentation>Сообщение об ошибке при отмена заказа.</Documentation>
      <Members>
        <Member>
          <Name>orderId</Name>
          <DataTypeName>Integer</DataTypeName>
        </Member>
        <Member>
          <Name>reason</Name>
```

```
          <DataTypeName>Message</DataTypeName>
        </Member>
      </Members>
    </Structure>
  </DataTypes>
  <Messages>
    <Message>
      <Name>ADD_STOCK_ORDER</Name>
      <MsgId>1</MsgId>
      <DataTypeName>AddStockOrderData</DataTypeName>
      <Senders>
        <ProcessRef>StockClient</ProcessRef>
      </Senders>
      <Receivers>
        <ProcessRef>StockServer</ProcessRef>
      </Receivers>
    </Message>
    <Message>
      <Name>ADD_STOCK_ORDER_ACK</Name>
      <MsgId>2</MsgId>
      <DataTypeName>AddStockOrderAckData</DataTypeName>
      <Senders>
        <ProcessRef>StockServer</ProcessRef>
      </Senders>
      <Receivers>
        <ProcessRef>StockClient</ProcessRef>
      </Receivers>
    </Message>
    <Message>
      <Name>ADD_STOCK_ORDER_NACK</Name>
      <MsgId>3</MsgId>
      <DataTypeName>AddStockOrderNackData</DataTypeName>
      <Senders>
        <ProcessRef>StockServer</ProcessRef>
      </Senders>
      <Receivers>
        <ProcessRef>StockClient</ProcessRef>
      </Receivers>
    </Message>
    <Message>
      <Name>CANCEL_STOCK_ORDER</Name>
      <MsgId>4</MsgId>
      <DataTypeName>CancelStockOrderData</DataTypeName>
      <Senders>
```

```
      <ProcessRef>StockClient</ProcessRef>
    </Senders>
    <Receivers>
      <ProcessRef>StockServer</ProcessRef>
    </Receivers>
  </Message>
  <Message>
    <Name>CANCEL_STOCK_ORDER_ACK</Name>
    <MsgId>5</MsgId>
    <DataTypeName>CancelStockOrderAckData</DataTypeName>
    <Senders>
      <ProcessRef>StockServer</ProcessRef>
    </Senders>
    <Receivers>
      <ProcessRef>StockClient</ProcessRef>
    </Receivers>
  </Message>
  <Message>
    <Name>CANCEL_STOCK_ORDER_NACK</Name>
    <MsgId>6</MsgId>
    <DataTypeName>CancelStockOrderNackData</DataTypeName>
    <Senders>
      <ProcessRef>StockServer</ProcessRef>
    </Senders>
    <Receivers>
      <ProcessRef>StockClient</ProcessRef>
    </Receivers>
  </Message>
  <Message>
    <Name>TEST</Name>
    <MsgId>7</MsgId>
    <DataTypeName>TestData</DataTypeName>
    <Senders>
      <ProcessRef>StockServer</ProcessRef>
    </Senders>
    <Receivers>
      <ProcessRef>StockClient</ProcessRef>
    </Receivers>
  </Message>
</Messages>
<Processes>
  <Process>
    <Name>StockClient</Name>
  </Process>
```

```
    <Process>
      <Name>StockServer</Name>
    </Process>
  </Processes>
</MessageRepository>
```

Этот репозиторий описывает сообщения, которыми обмениваются клиент (он называется `StockClient`) и сервер (`StockServer`) для выполнения различных операций. Читатели, знакомые с языком WSDL, отметят несомненное сходство; однако WSDL применяется для описания Web-сервисов и используется в контексте протокола SOAP, хотя технически его спецификация ни от какого протокола не зависит (*http://www.w3.org/TR/wsdl*).

Последние два примера в этой главе не связаны с задачей обмена сообщениями. Первый из них посвящен генерации кода на языке C++ из модели на унифицированном языке моделирования (UML), которая экспортируется инструментом разработки в виде файла на языке XMI (XML Metadata Interchange – обмен XML-метаданными). Во втором обсуждается применение XSLT для генерации XSLT.

Прежде чем переходить к самим примерам, хочу принести извинения за то, что в большинстве примеров используется язык C++. Я выбрал этот язык только потому, что хорошо с ним знаком; именно для него я написал реальные генераторы. Концептуально же весь каркас переносится и на другие языки, пусть даже фактический XSLT-код придется переделать[1].

XSLT 2.0

В силу особенностей рецептов ниже я не стал приводить решения на XSLT 1.0 и 2.0 по отдельности, как в большинстве других глав. Читателям, которым интересно, как применять для генерации кода XSLT 2.0, рекомендую обратиться в главе 6. Многие описанные там особенности XSLT 2.0 и способы их применения, подходят и для генерации кода. Вот несколько общих рекомендаций:

- Применяйте новый атрибут `separator` элемента `xsl:value-of`.
  При генерации кода часто приходится порождать последовательности элементов, разделенных какими-то символами (например, списки значений, разделенных запятыми). В таких случаях способность `xsl:value-of` автоматически вставлять разделитель может упростить генератор.
- Пользуйтесь возможностями последовательностей в XPath 2.0.
  Один из самых серьезных недостатков генераторов, написанных на XSLT 1.0, – это их громоздкость. В генераторах часто встречаются циклы и условные конструкции, а команды `xsl:for-each` и `xsl:choose` трудно назвать образцами краткости. Однако многие задачи генерации можно решить на XPath 2.0 вместо XSLT. Нередко XSLT и XPath применяются для

[1] Хочется добавить, пусть даже наполовину в шутку, что C++ – настолько сложный язык, что именно у программистов, работающих на нем, чаще всего возникает желание генерировать код, а не писать его вручную.

преобразования входного XML в одну или несколько последовательностей, после чего с помощью XPath они отображаются на код:

```
<!-- Функция для генерации объявления функции на языке C. -->
<!-- Используется преимущественно XPath 2.0 -->

<xsl:function name="ckbk:function-decl" as="xs:string*">
  <xsl:param name="name" as="xs:string"/>
  <xsl:param name="returnType" as="xs:string"/>
  <xsl:param name="argNames" as="xs:string*"/>
  <xsl:param name="argTypesPre" as="xs:string*"/>
  <xsl:param name="argTypesPost" as="xs:string*"/>
  <xsl:variable name="c" select="count($argNames)"/>
  <xsl:sequence select="$returnType,
                        $name,
                        '(',
                        for $i in 1 to $c return ($argTypesPre[$i],
                               $argNames[$i], $argTypesPost[$i],
                               if ($i ne $c) then ',' else ''),
                        ');' "/>
</xsl:function>
```

- Модульный, повторно используемый XSLT-код проще писать в версии 2.0. Если вы собрались писать генераторы, позволяющие транслировать XML-описание на различные языки, то стоит получше освоить приемы, описанные в рецептах 6.3 и 6.4.
- В версии 2.0 стандартизован вывод нескольких документов.
  Авторы генераторов для языков типа C и C++, где заголовочные и исходные файлы разделены, наверняка оценят удобство стандартизованной команды `xsl:result-document` для написания таблиц стилей, порождающих несколько выходных документов.

12.1. Генерация определений констант

Задача

Требуется сгенерировать исходный файл, содержащий имена всех сообщений в виде констант, причем значение константы должно быть равно идентификатору сообщения.

Решение

Можно построить преобразование на язык C++, которое легко адаптируется для языков C, C# и Java:

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

```
<xsl:output method="text"/>
<xsl:strip-space elements="*"/>

 <!-- Имя выходного заголовочного файла. -->
<xsl:param name="file" select=" 'MESSAGE_IDS.h' "/>

<!-- По умолчанию генерируются константы в синтаксисе C++ -->
<xsl:variable name="constants-type" select=" 'const int' "/>

<!-- По умолчанию используется оператор присваивания C++ -->
<xsl:variable name="assignment" select=" ' = ' "/>

<!-- По умолчанию используется символ конца предложения C++  -->
<xsl:variable name="terminator" select=" ';' "/>

<!-- Преобразовать репозиторий в последовательность определений
константа, соответствующих сообщениям -->
<xsl:template match="MessageRepository">
    <xsl:call-template name="constants-start"/>
    <xsl:apply-templates select="Messages/Message"/>
    <xsl:call-template name="constants-end"/>
</xsl:template>

<!-- Каждое сообщение превращается в комментарий и определение
константы -->
<xsl:template match="Message">
    <xsl:apply-templates select="." mode="doc" />
    <xsl:apply-templates select="." mode="constant" />
</xsl:template>

<!-- Заголовочные файлы в C++ начинаются с директив, предотвращающих -->
<!-- повторное включение (стражи) -->
<xsl:template name="constants-start">
    <xsl:variable name="guard" select="translate($file,'.','_')"/>
    <xsl:text>#ifndef </xsl:text>
    <xsl:value-of select="$guard"/>
    <xsl:text>&#xa;</xsl:text>
    <xsl:text>#define </xsl:text>
    <xsl:value-of select="$guard"/>
    <xsl:text>&#xa;&#xa;&#xa;</xsl:text>
</xsl:template>

<!-- Заголовочные файлы в C++ заканчиваются директивой, закрывающей -->
<!-- директиву-страж в начале -->
```

```
<xsl:template name="constants-end">
    <xsl:variable name="guard" select="translate($file,'.','_')"/>
    <xsl:text>&#xa;&#xa;&#xa;#endif /* </xsl:text>
    <xsl:value-of select="$guard"/>
    <xsl:text> */&#xa;</xsl:text>
</xsl:template>

<!-- Определению каждой константы предшествует комментарий, описывающий -->
<!-- соответствующее ей сообщение -->
<xsl:template match="Message" mode="doc">
/*
* Назначение: <xsl:call-template name="format-comment">
                <xsl:with-param name="text" select="Documentation"/>
                </xsl:call-template>
* Формат данных:  <xsl:value-of select="DataTypeName"/>
* Отправитель:    <xsl:apply-templates select="Senders" mode="doc"/>
* Получатель:     <xsl:apply-templates select="Receivers" mode="doc"/>
*/
</xsl:template>

<!-- Используется для генерации документации по сообщениям.
Перечисляет процессы-отправители и получатели -->
<xsl:template match="Senders|Receivers" mode="doc">
    <xsl:for-each select="ProcessRef">
        <xsl:value-of select="."/>
        <xsl:if test="position(  ) != last(  )">
            <xsl:text>, </xsl:text>
        </xsl:if>
    </xsl:for-each>
</xsl:template>

<!-- Этот шаблон разбивает комментарии на строки длиной 40 символов -->
<xsl:template name="format-comment">
  <xsl:param name="text"/>
  <xsl:choose>
    <xsl:when test="string-length($text)&lt;40">
      <xsl:value-of select="$text"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="substring($text,1,39)"/>
      <xsl:text>*&#xa;</xsl:text>
      <xsl:call-template name="format-comment">
        <xsl:with-param name="text" select="substring($text,40)"/>
```

```
        </xsl:call-template>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <!-- Каждое сообщение превращается в константу, значение которой
  совпадает с идентификатором сообщения -->
  <xsl:template match="Message" mode="constant">
      <xsl:value-of select="$constants-type"/><xsl:text> </xsl:text>
      <xsl:value-of select="Name"/>
      <xsl:value-of select="$assignment"/>
      <xsl:value-of select="MsgId"/>
      <xsl:value-of select="$terminator"/>
      <xsl:text>&#xa;</xsl:text>
  </xsl:template>

  <!-- Игнорировать текстовые узлы, не обработанные явно предыдущими
  шаблонами -->
  <xsl:template match="text(  )"/>

</xsl:stylesheet>
```

Применив эту таблицу стилей к нашему репозиторию, получим такой код:

```
#ifndef MESSAGE_IDS_h
#define MESSAGE_IDS_h

  /*
  * Назначение:     Запрос на добавление нового заказа.
  * Формат данных:  AddStockOrderData
  * Отправитель:    StockClient
  * Получатель:     StockServer
  */
  const int ADD_STOCK_ORDER_ID = 1;

  /*
  * Назначение:     Подтверждение успешного добавления
  *                 нового заказа.
  * Формат данных:  AddStockOrderAckData
  * Отправитель:    StockServer
  * Получатель:     StockClient
  */
  const int ADD_STOCK_ORDER_ACK_ID = 2;
```

```
  /*
  * Назначение:     Сообщение об ошибке при добавлении
  *                 нового заказа.
  * Формат данных:  AddStockOrderNackData
  * Отправитель:    StockServer
  * Получатель:     StockClient
  */
  const int ADD_STOCK_ORDER_NACK_ID = 3;

//И т.д. ...

#endif /* MESSAGE_IDS_h */
```

Обсуждение

Чтобы преобразование можно было адаптировать и для других языков, я сделал таблицу стилей более сложной, чем необходимо для одного языка. И все же в этой главе я не стал делать ее максимально общей. Например, комментарии выводятся в стиле, характерном для языков, берущих начало от C. Оформление комментариев тоже может кому-то не понравиться. Тем не менее, при создании собственных шаблонов для генерации кода стоит придерживаться следующих правил, обеспечивающих дальнейшую адаптацию:

- Кодируйте языково-зависимые конструкции в параметрах верхнего уровня или в переменных, чтобы их легко можно было переопределить в импортирующей таблице стилей или (если вы решите остановиться на параметрах) передать извне при запуске таблицы.
- Помещайте генерацию различных компонентов в отдельные шаблоны, которые можно будет переопределить в импортирующей таблице.

Спроектировав преобразование таким образом, легко с минимальными изменениями получить определение констант в стиле C в виде директив препроцессора `#define`:

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

  <xsl:import href="msgIds.xslt"/>

  <xsl:variable name="constants-type" select=" '#define ' "/>
  <xsl:variable name="assignment" select=" '    ' "/>
  <xsl:variable name="terminator" select=" '' "/>

</xsl:stylesheet>
```

В языке Java все определения должны находиться внутри класса, но и этого легко добиться:

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

```
<xsl:import href="msgIds.xslt"/>

<xsl:variable name="constants-type" select=" 'public static final int' "/>

  <xsl:template name="constants-start">
  <xsl:text>final public class MESSAGE_IDS &#xa;</xsl:text>
  <xsl:text>{&#xa;</xsl:text>
  </xsl:template>

  <xsl:template name="constants-end">
  <xsl:text>&#xa;&#xa;}&#xa;</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

12.2. Генерация предложения switch

Задача

Требуется сгенерировать предложение switch, которое будет передавать входящие сообщения подходящим обработчикам.

Решение

В репозитории сообщений хранится информация о том, какие процессы могут принимать те или иные сообщения. Поэтому, зная имя процесса, можно сгенерировать предложение switch для диспетчеризации входящих сообщений:

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

  <xsl:output method="text"/>

  <xsl:param name="process" select=" '*' "/>

  <xsl:variable name="message-dir" select=" 'messages' "/>
  <xsl:variable name="directory-sep" select=" '/' "/>
  <xsl:variable name="include-ext" select=" '.h' "/>

  <xsl:template match="MessageRepository">
      <!-- Генерируем пролог исходного файла -->
      <xsl:call-template name="file-start"/>

      <!-- Генерируем директивы include для включения файлов, -->
      <!-- в которых описываются сообщения, принимаемые этим процессом -->
      <xsl:apply-templates select="Messages/
```

```
                                    Message[Receivers/
                                            ProcessRef = $process or
                                              $process = '*']"
                            mode="includes"/>

      <!-- Генерируем пролог предложения switch -->
      <xsl:call-template name="switch-start"/>

      <!-- Генерируем тело предложения switch -->
      <xsl:apply-templates select="Messages/
                                    Message[Receivers/
                                            ProcessRef = $process or
                                              $process = '*']"
                            mode="switch"/>

      <!-- Генерируем эпилог предложения switch -->
      <xsl:call-template name="switch-end"/>

      <!-- Генерируем эпилог файла -->
      <xsl:call-template name="file-end"/>
   </xsl:template>

   <!-- Генерируем директивы include для отдельных сообщений -->
   <xsl:template match="Message" mode="includes">
      <xsl:text>#include &lt;</xsl:text>
      <xsl:value-of select="$message-dir"/>
      <xsl:value-of select="$directory-sep"/>
      <xsl:value-of select="Name"/>
      <xsl:value-of select="$include-ext"/>
      <xsl:text>&gt;&#xa;</xsl:text>
   </xsl:template>

   <!-- Генерируем ветвь case для одного типа сообщений -->
   <xsl:template match="Message" mode="switch">
      case <xsl:value-of select="Name"/>_ID:
         <xsl:call-template name="case-action"/>
   </xsl:template>

<!-- Генерируем действие - вызов обработчика сообщения -->
<xsl:template name="case-action">
      return <xsl:value-of select="Name"/>(*static_cast&lt;const
<xsl:value-of select="DataTypeName"/>*&gt;(msg.getData(  ))).process(  ) ;
</xsl:template>

   <!-- По умолчанию пустышки. При необходимости можно переопределить -->
```

```
  <xsl:template name="file-start"/>
  <xsl:template name="file-end"/>

  <!-- Генерируем пролог предложения switch -->
  <xsl:template name="switch-start">
#include &lt;transport/Message.h&gt;
#include &lt;transport/MESSAGE_IDS.h&gt;

<xsl:text>&#xa;&#xa;</xsl:text>
<xsl:call-template name="process-function"/>
{
  switch (msg.getId(  ))
  {
  </xsl:template>

  <xsl:template name="switch-end">
    return false ;
  }
}
  </xsl:template>

<!-- Генерируем сигнатуру точки входа в обработчик сообщения -->
<xsl:template name="process-function">
bool processMessage(const Message&amp;amp; msg)
</xsl:template>

</xsl:stylesheet>
```

Применив эту таблицу стилей к тестовому репозиторию, получим на выходе следующий код:

```
#include <messages/ADD_STOCK_ORDER.h>
#include <messages/CANCEL_STOCK_ORDER.h>

#include <transport/Message.h>
#include <transport/MESSAGE_IDS.h>

bool processMessage(const Message& msg)

{
  switch (msg.getId(  ))
  {

    case ADD_STOCK_ORDER_ID:

      return ADD_STOCK_ORDER(*static_cast<const
```

```
        AddStockOrderData*>(msg.getData( ))).process( ) ;

      case CANCEL_STOCK_ORDER_ID:

        return CANCEL_STOCK_ORDER(*static_cast<const
        CancelStockOrderData*>(msg.getData( ))).process( ) ;

      return false ;
    }
  }
```

Обсуждение

В приложениях, которые обрабатывают сообщения, всегда имеется какое-то ветвление по типу сообщений. Конструкции могут различаться, но как правило проверяется идентификатор сообщения, и в зависимости от его значения сообщение направляется тому или иному обработчику. Идентификатор может быть целым числом (как в данном случае) или строкой. Обработчик же может быть простой функцией или объектом, создаваемым специально для обработки сообщения. Даже совсем простые обработчики важно делать модульными, чтобы ими можно было воспользоваться в разных контекстах. В компании, где я когда-то работал, группа программистов решила создать очень интересную систему на языке Perl, которая генерировала бы C++-код из XSD-схемы. Она оказалась настолько полезной, что ей захотели пользоваться и другие группы. К сожалению, разработчики задумывали систему только для своих нужд, поэтому во всех остальных отношениях замечательное решение оказалось непригодным для повторного использования. Написание даже простейшего генератора кода – сложная задача, поэтому стремитесь сделать его максимально общим, чтобы другим не пришлось заново преодолевать все трудности.

В приведенном выше решении есть несколько мест, в которые можно вклиниться и изменить код, появляющийся в начале генерируемого файла, в начале и в конце предложения switch и в конце файла. Такая гибкость позволит нам повторно использовать этот генератор в рецепте 12.5. Но можно пойти и дальше. Например, в нашем генераторе предполагается, что ветвление по типу сообщения реализовано предложением switch из языка C. Однако в некоторых приложениях для этой цели применяется последовательность предложений if-else, особенно если идентификатор сообщения – строка. В других случаях используется таблица, сопоставляющая идентификатору сообщения указатель на функцию.

Можно ли написать единственный генератор, достаточно общий для обработки всех вариантов? Может быть, но, скорее всего, он окажется чрезвычайно сложным. Лучше создать компоненты, которыми можно пользоваться в сложных генераторах. Вот, например, обобщенный генератор предложения switch в стиле C:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
  <!-- Служит для управления отступами -->
  <!ENTITY INDENT "    ">
  <!ENTITY INDENT2 "&INDENT;&INDENT;">
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template name="gen-C-Switch">
    <xsl:param name="variable"/>
    <xsl:param name="cases" select="/.."/>
    <xsl:param name="actions" select="/.."/>
    <xsl:param name="default"/>
    <xsl:param name="baseIndent" select="'&INDENT;'"/>
    <xsl:param name="genBreak" select="true(  )"/>

    <xsl:value-of select="$baseIndent"/>
    <xsl:text>switch (</xsl:text>
    <xsl:value-of select="$variable"/>
    <xsl:text>)&#xa;</xsl:text>
    <xsl:value-of select="$baseIndent"/>
    <xsl:text>{&#xa;</xsl:text>

    <xsl:for-each select="$cases">
      <xsl:variable name="pos" select="position(  )"/>

      <xsl:value-of select="$baseIndent"/>
      <xsl:text>&INDENT;case </xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>:&#xa;</xsl:text>
      <xsl:call-template name="gen-C-Switch-caseBody">
        <xsl:with-param name="case" select="."/>
        <xsl:with-param name="action" select="$actions[$pos]"/>
        <xsl:with-param name="baseIndent"
                        select="concat('&INDENT2;',$baseIndent)"/>
      </xsl:call-template>
      <xsl:if test="$genBreak">
        <xsl:value-of select="$baseIndent"/>
        <xsl:text>&INDENT2;break;</xsl:text>
      </xsl:if>
      <xsl:text>&#xa;</xsl:text>
    </xsl:for-each>

    <xsl:if test="$default">
```

```
      <xsl:value-of select="$baseIndent"/>
      <xsl:text>&INDENT;default:</xsl:text>
      <xsl:text>:&#xa;</xsl:text>
      <xsl:call-template name="gen-C-Switch-default-caseBody">
        <xsl:with-param name="action" select="$default"/>
        <xsl:with-param name="baseIndent"
             select="concat('&INDENT2;',$baseIndent)"/>
      </xsl:call-template>
      <xsl:text>&#xa;</xsl:text>
    </xsl:if>
    <xsl:value-of select="$baseIndent"/>
    <xsl:text>}&#xa;</xsl:text>
  </xsl:template>

  <!-- По умолчанию генерирует пустое предложение. -->
  <!-- Переопределить для генерации кода ветви case. -->
  <xsl:template name="gen-C-Switch-caseBody">
    <xsl:param name="case"/>
    <xsl:param name="action"/>
    <xsl:param name="baseIndent"/>

    <xsl:value-of select="$baseIndent"/>
    <xsl:text>;</xsl:text>
  </xsl:template>

  <!-- Вызывает стандартный генератор тела ветви case. -->
  <!-- Переопределить, если нужно сделать что-то особое для -->
  <!-- ветви default. -->
  <xsl:template name="gen-C-Switch-default-caseBody">
    <xsl:param name="action"/>
    <xsl:param name="baseIndent"/>

    <xsl:call-template name="gen-C-Switch-caseBody">
      <xsl:with-param name="action" select="$action"/>
      <xsl:with-param name="baseIndent" select="$baseIndent"/>
    </xsl:call-template>
  </xsl:template>

</xsl:stylesheet>
```

В главе 14 обсуждается техника, которая позволяет сделать такие генераторы еще более общими.

12.3. Генерация кода заглушки обработчика сообщения

Задача

Требуется сгенерировать скелет обработчика сообщения.

Решение

Следующая таблица стилей создает простой скелет обработчика. Она принимает на входе имя процесса и генерирует код заглушки.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>

  <!-- Определяет, для какого процесса генерировать обработчики сообщений -->
  <xsl:param name="process"/>
  <!-- Определяет, для каких сообщений генерировать обработчики. -->
  <!-- Специальное значение %ALL% означает "для всех сообщений" -->
  <xsl:param name="message" select=" '%ALL%' "/>

   <!-- Каталог, в котором создаются выходные файлы -->
  <xsl:variable name="message-dir" select=" 'messages' "/>
  <xsl:variable name="directory-sep" select=" '/' "/>
  <xsl:variable name="include-ext" select=" '.h' "/>

  <xsl:template match="MessageRepository">
    <xsl:choose>
      <xsl:when test="$message='%ALL%'"  >
          <xsl:apply-templates
            select="Messages/Message[Receivers/ProcessRef = $process]"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates
          select="Messages/Message[Receivers/ProcessRef = $process and
                                   Name=$message]"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

    <xsl:template match="Message">
    <xsl:document href="{concat(Name,'.h')}">
      <xsl:call-template name="makeHeader"/>
```

```
    </xsl:document>
    <xsl:document href="{concat(Name,'.cpp')}">
      <xsl:call-template name="makeSource"/>
    </xsl:document>
    </xsl:template>

    <xsl:template name="makeHeader">
#ifndef <xsl:value-of select="Name"/>_h
#define <xsl:value-of select="Name"/>_h

#include &lt;transport/MessageHandler.h&gt;

// Опережающие объявления
class <xsl:value-of select="DataTypeName"/> ;
/*!TODO:  Вставить сюда дополнительные опережающие объявления.*/

class <xsl:value-of select="Name"/> : public MessageHandler
{
public:
  <xsl:value-of select="Name"/>(const <xsl:value-of
     select="DataTypeName"/>&amp; data) ;
  bool process(  ) ;
private:

  const <xsl:value-of select="DataTypeName"/>&amp; m_Data ;
} ;

#endif
  </xsl:template>

  <xsl:template name="makeSource">
#include &lt;messages/<xsl:value-of select="Name"/>.h&gt;

/*!TODO:  Вставить сюда дополнительные директивы include.*/

<xsl:value-of select="Name"/>::<xsl:value-of select="Name"/>(const
    <xsl:value-of select="DataTypeName"/>&amp; data)
  : m_Data(data)
{
}
bool <xsl:value-of select="Name"/>::process(  )
{
  /*!TODO:  Вставить сюда код обработчика. */
  return true;
}
```

```
  </xsl:template>

</xsl:stylesheet>
```

Эта таблица стилей генерирует заголовочный и исходный файл для каждого обрабатываемого ей сообщения. Результат показан в примерах 12.1 и 12.2.

Пример 12.1. AddStockOrder.h

```
#ifndef ADD_STOCK_ORDER_h
#define ADD_STOCK_ORDER_h

#include <transport/MessageHandler.h>

// Опережающие объявления
class AddStockOrderData ;
/*!TODO:  Вставить сюда дополнительные опережающие объявления.*/

class ADD_STOCK_ORDER : public MessageHandler
{
public:
     ADD_STOCK_ORDER(const AddStockOrderData& data) ;
     bool process(  ) ;
private:

     const AddStockOrderData& m_Data ;
} ;

#endif
```

Пример 12.2. AddStockOrder.cpp

```
#include <messages/ADD_STOCK_ORDER.h>

/*!TODO:  Вставить сюда дополнительные директивы include.*/

ADD_STOCK_ORDER::ADD_STOCK_ORDER(const AddStockOrderData& data)
  : m_Data(data)
{
}

bool ADD_STOCK_ORDER::process(  )
{
  /*!TODO:  Вставить сюда код обработчика. */
  return true;
}
```

Обсуждение

Разработчикам часто приходится писать фрагменты кода, которые имеют одинаковую структуру, но разное содержание. Иными словами, мы пишем один и тот же шаблонный код, специализированный для конкретного контекста. Это скучно. А, когда скучно, рассеивается внимание. А это уже ведет к ошибкам. Сгенерировав повторяющиеся части кода автоматически, вы сможете сосредоточиться на действительно важных вещах.

Инструменты, генерирующие код с участками TODO (СДЕЛАТЬ), часто называют мастерами. Приведенный выше мастер генерации обработчиков сообщений – очень простой пример такой программы. Некоторые коммерческие мастера генерируют структуру всего приложения. Неясно, потянет ли XSLT создание мастеров такого масштаба. Но для простых генераторов заглушек в случае, когда исходные данные представлены в формате XML, XSLT лучше многих других языков, включая и Perl. Многие фанатики Perl не согласятся с последним утверждением, так как обработку XML они рассматривают просто как разновидность обработки текста (а всем известно, что язык Perl удерживает неоспоримое первенство в этой области). Однако можно привести веские аргументы в пользу того, обработка XML – это не столько обработка текста, сколько деревьев, содержащих текстовые узлы. А с преобразованиями деревьев XSLT справляется куда ловчее, чем Perl. Но Perl и XSLT можно использовать и совместно, взяв лучшее от того и другого. Такой подход описан в главе 14. Там же представлено расширение XSLT, с помощью которого можно устранить многословность, свойственную генераторам кода на чистом XSLT.

12.4. Генерация оберток для данных

Задача

Требуется создать классы, которые надстраивали бы над заключенными в сообщение данными интерфейс, безопасный с точки зрения типов.

Решение

Показанная ниже таблица стилей работает в двух режимах. Если в качестве параметра передано имя сообщения, то генерируется обертка только для этого сообщения. В противном случае генерируются обертки для всех сообщений.

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <!-- Сообщение, для которого генерировать обертку. '*' – для всех -->
  <xsl:param name="message" select=" '*' "/>
  <!-- Каталог, куда помещать сгенерированные файлы -->
  <xsl:param name="generationDir" select=" 'src/' "/>
```

```
    <!-- Расширение имени заголовочного файла C++ -->
    <xsl:param name="headerExt" select=" '.h' "/>
    <!-- Расширение имени исходного файла C++ -->
    <xsl:param name="sourceExt" select=" '.C' "/>

    <!-- Ключ для поиска типов данных по имени -->
    <xsl:key name="dataTypes" match="Structure" use="Name" />
    <xsl:key name="dataTypes" match="Primitive" use="Name" />
    <xsl:key name="dataTypes" match="Array" use="Name" />
    <xsl:key name="dataTypes" match="Enumeration" use="Name" />

    <!-- Шаблон верхнего уровня определяет, какие сообщения обрабатывать -->
    <xsl:template match="/">
      <xsl:choose>
          <xsl:when test="$message = '*'">
            <xsl:apply-templates select="*/Messages/*"/>
          </xsl:when>
          <xsl:when test="*/Messages/Message[Name=$message]">
            <xsl:apply-templates select="*/Messages/Message[Name=$message]"/>
        </xsl:when>
          <xsl:otherwise>
            <xsl:message terminate="yes">No such message name
            [<xsl:value-of select="$message"/>]</xsl:message>
          </xsl:otherwise>
       </xsl:choose>
    </xsl:template>

<!-- Если тип данных, представленный сообщением, описан в репозитории,
сгенерировать для него заголовочный и исходный файл -->
<xsl:template match="Message">
  <xsl:choose>
    <xsl:when test="key('dataTypes',DataTypeName)">
      <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                           mode="header"/>
      <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                           mode="source"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message>Message name [<xsl:value-of select="Name"/>] uses data
       [<xsl:value-of select="DataTypeName"/>] that is not defined in the
         repository.</xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

```
<!-- Заголовки генерируются только, если тип данных в сообщении -
Structure. Возможен также тип данных XML. Если полезная нагрузка
представлена в формате XML, обертка не генерируется. -->
<xsl:template match="Structure" mode="header">
<xsl:document href="{concat($generationDir,Name,$headerExt)}">
#include &lt;primitives/primitives.h&gt;

class <xsl:value-of select="Name"/>
{
public:<xsl:text>&#xa;&#xa;</xsl:text>
  <xsl:for-each select="Members/Member">
    <xsl:text>    </xsl:text>
    <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                         mode="returnType"/>
    get_<xsl:value-of select="Name"/>(  ) const ;<xsl:text/>
    <xsl:text>&#xa;</xsl:text>
   </xsl:for-each>
<xsl:text>&#xa;</xsl:text>
private:<xsl:text>&#xa;&#xa;</xsl:text>
  <xsl:for-each select="Members/Member">
    <xsl:text>    </xsl:text>
    <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                         mode="data"/>  m_
<xsl:value-of select="Name"/> ;<xsl:text/>
    <xsl:text>&#xa;</xsl:text>
  </xsl:for-each>
} ;
</xsl:document>
</xsl:template>

<!-- Исходные файлы генерируются только, если тип данных в сообщении -
Structure. Возможен также тип данных XML. Если полезная нагрузка
представлена в формате XML, обертка не генерируется. -->
<xsl:template match="Structure" mode="source">
<xsl:document href="{concat($generationDir,Name,$sourceExt)}">
#include "<xsl:value-of select="Name"/><xsl:value-of
select="$headerExt"/>"

<xsl:text/>

  <xsl:for-each select="Members/Member">
      <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                           mode="returnType"/>
      <xsl:text>  </xsl:text>
```

```
      <xsl:value-of select="../../Name"/>::get_<xsl:value-of
                          select="Name"/>(  ) const
      <xsl:text>&#xa;</xsl:text>
      <xsl:text>{&#xa;</xsl:text>
      <xsl:text>    return m_</xsl:text><xsl:value-of select="Name"/>
      <xsl:text>;&#xa;</xsl:text>
      <xsl:text>}&#xa;&#xa;</xsl:text>
  </xsl:for-each>

</xsl:document>
</xsl:template>

<!-- Предполагается, что члены, которые сами являются структурами, -->
<!-- возвращаются по ссылке. -->
<xsl:template match="Structure" mode="returnType">
const <xsl:value-of select="Name"/>&amp;<xsl:text/>
</xsl:template>

<!-- Примитивы, которые можно представить встроенными в C++ типами, -->
<!-- отображаются на эти типы. В противном случае предполагаем, что -->
<!-- примитив определен во внешнем файле. -->
<xsl:template match="Primitive" mode="returnType">
  <xsl:choose>
      <xsl:when test="Name='Integer' ">int</xsl:when>
      <xsl:when test="Name='Real' ">double</xsl:when>
      <xsl:otherwise><xsl:value-of select="Name"/></xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="*" mode="returnType">
<xsl:value-of select="Name"/>
</xsl:template>

<xsl:template match="Primitive" mode="data">
  <xsl:choose>
      <xsl:when test="Name='Integer' ">int</xsl:when>
      <xsl:when test="Name='Real' ">double</xsl:when>
      <xsl:otherwise><xsl:value-of select="Name"/></xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="*" mode="data">
<xsl:value-of select="Name"/>
```

```
</xsl:template>

</xsl:stylesheet>
```

Этот генератор порождает только методы `get`, но при необходимости его легко можно расширить так, что будут создавать методы `set` и любые другие. Вот пример сгенерированного заголовочного файла:

```
#include <primitives/primitives.h>

class AddStockOrderData
{
public:

      StkSymbol get_symbol(  ) const ;
      Shares get_quantity(  ) const ;
      BuyOrSell get_side(  ) const ;
      OrderType get_type(  ) const ;
      double get_price(  ) const ;

private:

      StkSymbol  m_symbol ;
      Shares  m_quantity ;
      BuyOrSell  m_side ;
      OrderType  m_type ;
      double  m_price ;

} ;
```

А вот как выглядит cpp-файл:

```
#include "AddStockOrderData.h"

StkSymbol  AddStockOrderData::get_symbol(  ) const
{
      return m_symbol;
}

Shares  AddStockOrderData::get_quantity(  ) const
{
      return m_quantity;
}

BuyOrSell  AddStockOrderData::get_side(  ) const
{
      return m_side;
```

```
}

OrderType  AddStockOrderData::get_type(  ) const
{
      return m_type;
}

double  AddStockOrderData::get_price(  ) const
{
      return m_price;
}
```

Обсуждение

В этом разделе термином *обертка* обозначается класс, предоставляющий объектно-ориентированный интерфейс к данным, которые на языке C записываются в виде структуры struct. Когда-то мне довелось работать над проектом, где все обертки сообщений кодировались вручную. Работа была утомительной, но результат того стоил. Возьмем, к примеру, сообщение, содержащее цену, количество и дату. Все эти высокоуровневые типы можно представить типом `int`. Если по ошибке использовать в программе одно вместо другого, компилятор ничего не заметит. Обертки позволяют надстроить над данными сообщения тонкий промежуточный слой, который преобразует низкоуровневое представление в примитивы типа класса, например: `Price`, `Qty` и `Date`. Автоматическая генерация оберток дает возможность сделать это без лишних усилий.

Репозиторий сообщений и XSLT-генератор позволяют автоматизировать процесс создания оберток. На практике обертки иногда содержат дополнительную логику, для генерации которой нужно включить в репозиторий добавочные метаданные. Типичный случай возникает, когда данные сообщения содержат массивы. Часто имеется еще специальное поле, в котором хранится количество элементов в массиве. Если в обертку вручную включается код добавления элемента в массив, то придется взять значение этого поля, которое дает индекс следующей незаполненной ячейки, и увеличить его после помещения новых данных. Сгенерировать такой год можно лишь в том случае, если в репозитории есть информация о том, с каким массивом ассоциировано поле размера.

12.5. Генерация функций форматированной печати

Задача

Необходимы инструменты для отладки приложения. В частности, хотелось бы иметь возможность распечатывать двоичные сообщения в форме, понятной человеку.

Решение

При разработке приложений, обрабатывающих сообщения, программисты часто вручную пишут функции для их распечатки, поскольку это заметно упрощает отладку. Однако при наличии репозитория сообщений такой код можно сгенерировать автоматически. В этом решении показано, как можно повторно использовать генератор предложения switch из рецепта 12.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
   <!-- Служит для управления отступами -->
   <!ENTITY INDENT "    ">
   <!ENTITY INDENT2 "&INDENT;&INDENT;">
   <!ENTITY LS "&lt;&lt;">
]>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Для этого генератора функций печати нужно ветвление по типу сообщения, -->
<!-- поэтому повторно используем написанное ранее. -->
<xsl:import href="messageSwitch.xslt"/>

  <!-- Каталог, в котором хранятся сгенерированные файлы -->
  <xsl:param name="generationDir" select=" 'src/' "/>
  <!-- Имя заголовочного файла C++ -->
  <xsl:param name="prettyPrintHeader" select=" 'prettyPrint.h' "/>
  <!-- Имя исходного файла C++ -->
  <xsl:param name="prettyPrintSource" select=" 'prettyPrint.C' "/>

  <!-- Ключ для поиска типов данных по имени -->
  <xsl:key name="dataTypes" match="Structure" use="Name" />
  <xsl:key name="dataTypes" match="Primitive" use="Name" />
  <xsl:key name="dataTypes" match="Array" use="Name" />
  <xsl:key name="dataTypes" match="Enumeration" use="Name" />

  <xsl:template match="MessageRepository">
    <xsl:document href="{concat($generationDir,$prettyPrintHeader)}">
      <xsl:text>void prettyPrintMessage</xsl:text>
     <xsl:text>(ostream&amp; stream, const Message&amp; msg);&#xa;</xsl:text>
      <xsl:apply-templates select="DataTypes/Structure" mode="declare"/>
    </xsl:document>

    <xsl:document href="{concat($generationDir,$prettyPrintSource)}">
      <xsl:apply-imports/>
      <xsl:apply-templates select="DataTypes/Structure" mode="printers"/>
    </xsl:document>
```

```
  </xsl:template>

<!-- Переопределяем имя функции обработки сообщения в шаблоне -->
<!-- из таблицы messageSwitch.xslt, так что теперь в ее сигнатуре -->
<!-- указано, что на вход подается поток -->
<xsl:template name="process-function">
<xsl:text>void prettyPrintMessage</xsl:text>
<xsl:text>(ostream&amp;amp; stream, const Message&amp;amp; msg)</xsl:text>
</xsl:template>

<!-- Переопределяем действие в ветви case в шаблоне из таблицы -->
<!-- messageSwitch.xslt, так чтобы вызывалась функция prettyPrinter -->
<xsl:template name="case-action">
 <xsl:text>   prettyPrint(stream, *static_cast&lt;const </xsl:text>
 <xsl:value-of select="DataTypeName"/>
 <xsl:text>*&gt;(msg.getData(  ))) ;
        break;</xsl:text>
</xsl:template>

<!-- Генерируем объявления для каждого типа сообщений -->
<xsl:template match="Structure" mode="declare">
<!-- Опережающее объявление класса, представляющего данные сообщения  -->
<xsl:text>class </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text> ;&#xa;</xsl:text>
<!-- Опережающее объявление функции prettyPrint для печати сообщения -->
<xsl:text>ostream prettyPrint(ostream &amp;amp; stream, const </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>&amp;amp; data);&#xa;</xsl:text>
</xsl:template>

<!-- Генерируем тело функции печати -->
<xsl:template match="Structure" mode="printers">
<xsl:text>ostream prettyPrint(ostream &amp;amp; stream, const </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>&amp;amp; data)&#xa;</xsl:text>
<xsl:text>{&#xa;</xsl:text>
<xsl:text>&INDENT;stream &#xa;</xsl:text>
<xsl:text>&INDENT2;&LS; "</xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>" &LS;  endl  &LS; "{"  &LS; endl &#xa;</xsl:text>
  <xsl:for-each select="Members/Member">
      <xsl:text>&INDENT2;&LS; "</xsl:text>
      <xsl:value-of select="Name"/>: " &LS; <xsl:text/>
      <xsl:apply-templates
```

```
                    select="key('dataTypes',DataTypeName)" mode="print">
          <xsl:with-param name="name" select="Name"/>
        </xsl:apply-templates>
        <xsl:text>&#xa;</xsl:text>
    </xsl:for-each>
    <xsl:text>&INDENT2;&LS; "}"  &LS; endl ; &#xa;</xsl:text>
    <xsl:text>&INDENT;return stream ;&#xa;</xsl:text>
    <xsl:text>}&#xa;&#xa;</xsl:text>
</xsl:template>

<!-- При наличии вложенной структуры для нее вызывается функция печати -->
<xsl:template match="Structure" mode="print">
    <xsl:param name="name"/>
    <xsl:text>prettyPrint(stream, data.get_</xsl:text>
    <xsl:value-of select="$name"/><xsl:text>(  ))</xsl:text>
</xsl:template>

<!-- Предполагается, что для каждого примитивного компонента сообщения -->
<!-- имеется метод get -->
<xsl:template match="*" mode="print">
    <xsl:param name="name"/>
    <xsl:text>data.get_</xsl:text>
    <xsl:value-of select="$name"/>(  ) &lt;&lt; endl<xsl:text/>
</xsl:template>

</xsl:stylesheet>
```

В результате генерируется следующий исходный файл. Заголовочный файл мы опустили, так как он содержит только объявления.

```
#include <messages/ADD_STOCK_ORDER.h>
#include <messages/ADD_STOCK_ORDER_ACK.h>
#include <messages/ADD_STOCK_ORDER_NACK.h>
#include <messages/CANCEL_STOCK_ORDER.h>
#include <messages/CANCEL_STOCK_ORDER_ACK.h>
#include <messages/CANCEL_STOCK_ORDER_NACK.h>
#include <messages/TEST.h>

#include <transport/Message.h>
#include <transport/MESSAGE_IDS.h>

void prettyPrintMessage(ostream& stream, const Message& msg)
{
    switch (msg.getId(  ))
    {
```

```
      case ADD_STOCK_ORDER_ID:
           prettyPrint(stream, *static_cast<const
           AddStockOrderData*>(msg.getData(  ))) ;
           break;
      case ADD_STOCK_ORDER_ACK_ID:
           prettyPrint(stream, *static_cast<const
           AddStockOrderAckData*>(msg.getData(  ))) ;
           break;
      case ADD_STOCK_ORDER_NACK_ID:
           prettyPrint(stream, *static_cast<const
           AddStockOrderNackData*>(msg.getData(  ))) ;
           break;
      case CANCEL_STOCK_ORDER_ID:
           prettyPrint(stream, *static_cast<const
           CancelStockOrderData*>(msg.getData(  ))) ;
           break;
      case CANCEL_STOCK_ORDER_ACK_ID:
           prettyPrint(stream, *static_cast<const
           CancelStockOrderAckData*>(msg.getData(  ))) ;
           break;
      case CANCEL_STOCK_ORDER_NACK_ID:
           prettyPrint(stream, *static_cast<const
           CancelStockOrderNackData*>(msg.getData(  ))) ;
           break;
      case TEST_ID:
           prettyPrint(stream, *static_cast<const TestData*>(msg.getData( ))) ;
            break;
      return false ;
    }
}
   ostream prettyPrint(ostream & stream, const TestData& data)
{
    stream
      << "TestData" <<  endl  << "{"  << endl
      << "order: " << prettyPrint(stream, data.get_order(  ))
      << "cancel: " << prettyPrint(stream, data.get_cancel(  ))
      << "}"  << endl ;
    return stream ;
}

ostream prettyPrint(ostream & stream, const AddStockOrderData& data)
{
   stream
      << "AddStockOrderData" <<  endl  << "{"  << endl
```

```
      << "symbol: " << data.get_symbol(  ) << endl
      << "quantity: " << data.get_quantity(  ) << endl
      << "side: " << data.get_side(  ) << endl
      << "type: " << data.get_type(  ) << endl
      << "price: " << data.get_price(  ) << endl
      << "}"  << endl ;
   return stream ;
}

ostream prettyPrint(ostream & stream, const AddStockOrderAckData& data)
{
   stream
      << "AddStockOrderAckData" <<  endl  << "{"  << endl
      << "orderId: " << data.get_orderId(  ) << endl
      << "}"  << endl ;
   return stream ;
}

ostream prettyPrint(ostream & stream, const AddStockOrderNackData& data)
{
   stream
      << "AddStockOrderNackData" <<  endl  << "{"  << endl
      << "reason: " << data.get_reason(  ) << endl
      << "}"  << endl ;
   return stream ;
}

ostream prettyPrint(ostream & stream, const CancelStockOrderData& data)
{
   stream
      << "CancelStockOrderData" <<  endl  << "{"  << endl
      << "orderId: " << data.get_orderId(  ) << endl
      << "quantity: " << data.get_quantity(  ) << endl
      << "}"  << endl ;
   return stream ;
}

ostream prettyPrint(ostream & stream, const CancelStockOrderAckData& data)
{
   stream
      << "CancelStockOrderAckData" <<  endl  << "{"  << endl
      << "orderId: " << data.get_orderId(  ) << endl
      << "quantityRemaining: " << data.get_quantityRemaining(  ) << endl
      << "}"  << endl ;
```

```
  return stream ;
}

ostream prettyPrint(ostream & stream, const CancelStockOrderNackData& data)
{
  stream
     << "CancelStockOrderNackData" <<  endl  << "{"  << endl
     << "orderId: " << data.get_orderId(  ) << endl
     << "reason: " << data.get_reason(  ) << endl
     << "}"  << endl ;
  return stream ;
}
```

Обсуждение

В этом рецепте мы генерируем код печати для каждого сообщения. Понять, как это делается, несложно, а результирующий код достаточно эффективен. Но можно было бы подойти к решению задачи с более общих позиций и создать более полезный генератор кода.

Точнее, процесс печати можно разбить на два этапа. На первом этапе монолитное сообщение разбивается на составные части. На втором этапе каждая часть преобразуется в понятный человеку текст.

Если взглянуть на задачу под этим углом зрения, то вместо генерации набора функций для выполнения одной узкой задачи (форматированной печати) можно сгенерировать более общий анализатор сообщений. Обычно такие анализаторы управляются событиями. Читатель, знакомый с интерфейсом SAX (Simple API for XML), легко опознает знакомый стиль обработки. Таблица стилей для генерации анализатора сообщений – это вариация на тему генератора функций печати. Но вместо того, чтобы отправлять компоненты сообщения в поток, она передает события разбора обработчику:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
  <!-- Служит для управления отступами -->
  <!ENTITY INDENT "    ">
  <!ENTITY INDENT2 "&INDENT;&INDENT;">
  <!ENTITY LS "&lt;&lt;">
]>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- Для этого генератора программы разбора сообщений нужно ветвление -->
<!-- по типу сообщения, поэтому повторно используем написанное ранее. -->
<xsl:import href="messageSwitch.xslt"/>

  <!-- Каталог, в котором хранятся сгенерированные файлы -->
```

```
  <xsl:param name="generationDir" select=" 'src/' "/>
  <!-- Имя заголовочного файла C++ -->
  <xsl:param name="msgParseHeader" select=" 'msgParse.h' "/>
  <!-- Имя исходного файла C++ -->
  <xsl:param name="msgParseSource" select=" 'msgParse.C' "/>

  <!-- Ключ для поиска типов данных по имени -->
  <xsl:key name="dataTypes" match="Structure" use="Name" />
  <xsl:key name="dataTypes" match="Primitive" use="Name" />
  <xsl:key name="dataTypes" match="Array" use="Name" />
  <xsl:key name="dataTypes" match="Enumeration" use="Name" />

  <xsl:template match="MessageRepository">
    <xsl:document href="{concat($generationDir,$msgParseHeader)}">
      <xsl:text>void parseMessage</xsl:text>
       <xsl:text>(MessageHandler&amp; handler, const Message&amp; msg);&#xa;
       </xsl:text>
      <xsl:apply-templates select="DataTypes/Structure" mode="declare"/>
    </xsl:document>

    <xsl:document href="{concat($generationDir,$msgParseSource)}">
      <xsl:apply-imports/>
      <xsl:apply-templates select="DataTypes/Structure" mode="parsers"/>
    </xsl:document>

  </xsl:template>

<!-- Переопределяем имя функции обработки сообщения в шаблоне -->
<!-- из таблицы messageSwitch.xslt, так что теперь в ее сигнатуре -->
<!-- указано, что ей передается обработчик -->
<xsl:template name="process-function">
<xsl:text>void parseMessage</xsl:text>
<xsl:text>(MessageHandler&amp; handler, const Message&amp; msg)</xsl:text>
</xsl:template>

<!-- Переопределяем действие в ветви case в шаблоне из таблицы -->
<!-- messageSwitch.xslt, так чтобы вызывалась функция разбора -->
<!-- данных сообщения -->
<xsl:template name="case-action">
 <xsl:text>   parse(handler, *static_cast&lt;const </xsl:text>
 <xsl:value-of select="DataTypeName"/>
 <xsl:text>*&gt;(msg.getData(  ))) ;
         break;</xsl:text>
</xsl:template>
```

```
<!-- Генерируем объявления для каждого типа сообщений -->
<xsl:template match="Structure" mode="declare">
<!-- Опережающее объявление класса, представляющего данные сообщения  -->
<xsl:text>class </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text> ;&#xa;</xsl:text>
<!-- Опережающее объявление функции разбора сообщения -->
<xsl:text>void parse(MessageHandler &amp; handler, const </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>&amp; data);&#xa;</xsl:text>
</xsl:template>

<!-- Генерируем тело функции разбора сообщения -->
<xsl:template match="Structure" mode="parsers">
<xsl:text>void parse(MessageHandler &amp; handler, const </xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>&amp; data)&#xa;</xsl:text>
<xsl:text>{&#xa;</xsl:text>
<xsl:text>&INDENT;handler.beginStruct("</xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>") ;&#xa;</xsl:text>
  <xsl:for-each select="Members/Member">
      <xsl:apply-templates
           select="key('dataTypes',DataTypeName)" mode="parse">
        <xsl:with-param name="name" select="Name"/>
      </xsl:apply-templates>
  </xsl:for-each>
<xsl:text>&INDENT;handler.endStruct("</xsl:text>
<xsl:value-of select="Name"/>
<xsl:text>") ;&#xa;</xsl:text>
  <xsl:text>}&#xa;&#xa;</xsl:text>
</xsl:template>

<!-- При наличии вложенной структуры для нее вызывается функция разбора -->
<xsl:template match="Structure" mode="parse">
  <xsl:param name="name"/>
  <xsl:text>&INDENT;parse(handler, data.get_</xsl:text>
  <xsl:value-of select="$name"/><xsl:text>(  ));&#xa;</xsl:text>
</xsl:template>

<!-- Предполагается, что для каждого примитивного компонента сообщения -->
<!-- имеется метод get -->
<xsl:template match="*" mode="parse">
  <xsl:param name="name"/>
```

```
  <xsl:text>&INDENT;handler.field("</xsl:text>
  <xsl:value-of select="$name"/>","<xsl:text/>
  <xsl:value-of select="Name"/>",<xsl:text/>
  <xsl:text>data.get_</xsl:text>
  <xsl:value-of select="$name"/>(  )<xsl:text/>
  <xsl:text>);&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

В результате генерируются функции разбора такого вида:

```
void parse(MessageHandler & handler, const AddStockOrderData& data)
{
      handler.beginStruct("AddStockOrderData") ;
      handler.field("symbol","StkSymbol",data.get_symbol(  ));
      handler.field("quantity","Shares",data.get_quantity(  ));
      handler.field("side","BuyOrSell",data.get_side(  ));
      handler.field("type","OrderType",data.get_type(  ));
      handler.field("price","Real",data.get_price(  ));
      handler.endStruct("AddStockOrderData") ;
}
```

12.6. Генерация Web-клиента для тестирования ввода данных

Задача

Требуется автономно протестировать процесс путем ввода данных в форму с последующим преобразованием их в сообщение, которое посылается процессу.

Решение

В этом примере мы сгенерируем инструмент для ввода данных на стороне клиента. Это будет HTML-форма, в которой можно заполнить поля, составляющие сообщение. В форме указано, что данные должны обрабатываться CGI-сценарием, который мы сгенерируем в рецепте 12.7.

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

  <xsl:param name="message"/>

   <!-- Ключ для поиска типов данных по имени -->
  <xsl:key name="dataTypes" match="Structure" use="Name" />
  <xsl:key name="dataTypes" match="Primitive" use="Name" />
```

```
    <xsl:key name="dataTypes" match="Array" use="Name" />
    <xsl:key name="dataTypes" match="Enumeration" use="Name" />

    <xsl:template match="/">
       <html>
          <head>
             <title><xsl:value-of select="$message"/>Ввод данных</title>
          </head>
          <body bgcolor="#FFFFFF" text="#000000">
             <h1><xsl:value-of select="$message"/>Ввод данных</h1>
       <form name="{concat($message,'Form')}" method="post"
       action="{concat('/cgi-bin/',$message,'Process.pl')}">
          <xsl:apply-templates select="*/Messages/Message[Name=$message]"/>
          <br/><center><input type="submit" name="Submit" value="Submit"/>
                </center>
          </form>
          </body>
       </html>
    </xsl:template>

    <xsl:template match="Message">
       <xsl:apply-templates select="key('dataTypes',DataTypeName)">
          <xsl:with-param name="field" select="Name"/>
       </xsl:apply-templates>
    </xsl:template>

<xsl:template match="Structure">
  <xsl:param name="field"/>
  <table width="100%" border="0" cellspacing="1" cellpadding="1">
    <tbody>
      <xsl:for-each select="Members/Member">
        <tr>
          <td valign="top"><xsl:value-of select="Name"/></td>
          <td>
            <xsl:apply-templates select="key('dataTypes',DataTypeName)">
              <xsl:with-param name="field"
                        select="concat($field,'_',Name)"/>
            </xsl:apply-templates>
          </td>
        </tr>
      </xsl:for-each>
    </tbody>
  </table>
</xsl:template>
```

```
<xsl:template match="*">
  <xsl:param name="field"/>
  <input type="text" name="{$field}" size="30"/>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

Генерация пользовательского HTML-интерфейса – один из самых простых способов автоматически сгенерировать клиента для подачи тестовых данных. Но есть и другие. Например, можно было сгенерировать клиента, который выводит имя поля на `stdout` и читает его значение из `stdin`. Если вы чувствуете в себе силы, можете сгенерировать и клиента с графическим интерфейсом. К достоинствам решений, отличных от HTML, следует отнести тот факт, что функциональность этого рецепта и рецепта 12.7 можно было бы объединить в одном приложении. Но не удивляйтесь, если такой генератор окажется сложнее приводимых ниже; ведь подход на базе связки HTML-CGI опирается на уже готовую инфраструктуру: браузер и Web-сервер.

Существенным дополнением к этому рецепту стала бы генерация кода контроля данных на JavaScript или VBScript. Качество такого кода зависит от того, какие метаданные хранятся в репозитории. Можно было включить в него минимальные и максимальные допустимые значения или регулярные выражения для контроля данных.

12.7. Генерация CGI-сценария для обработки тестовых данных

Задача

Требуется автономно протестировать процесс путем обработки введенных в форму данных и преобразования их в физическое сообщение, которое можно послать тестируемому процессу.

Решение

Этот генератор сервера является дополнением к генератору клиента из рецепта 12.6. Сейчас мы приведем лишь часть решения, а детали отложим до раздела «Обсуждение».

Сгенерированный сервер – это CGI-программа на языке C++. Так приходится поступать, потому что введенные в форму данные преобразуются в двоичное сообщение в формате, определяемом моделью памяти, характерной для C. Предполагается, что написанные на C++ процессы, потребляющие сообщения, ожидают именно двоичных данных. В сгенерированной программе используется библиотека libcgi компании Enterprise Integration Technologies (*http://www.landfield.com/*

hypermail/source/libcgi/), которая решает типичные для CGI-программ задачи, к примеру, разбор строки запроса.

```
<!DOCTYPE xslt [
  <!-- Служит для управления отступами -->
  <!ENTITY INDENT "    ">
]>
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:output method="text"/>
  <xsl:strip-space elements="*"/>

  <!-- Для какого сообщения генерировать данные. '*' - для всех -->
  <xsl:param name="message" select=" '*' "/>
   <!-- Каталог, в котором хранятся сгенерированные файлы -->
  <xsl:param name="generationDir" select=" 'src/' "/>
  <!-- Расширение заголовочных файлов C++ -->
   <xsl:param name="headerExt" select=" '.h' "/>
   <!-- Расширение исходных файлов C++ -->
  <xsl:param name="sourceExt" select=" '.C' "/>

   <!-- Ключ для поиска типов данных по имени -->
  <xsl:key name="dataTypes" match="Structure" use="Name" />
  <xsl:key name="dataTypes" match="Primitive" use="Name" />
  <xsl:key name="dataTypes" match="Array" use="Name" />
  <xsl:key name="dataTypes" match="Enumeration" use="Name" />

 <!-- Шаблон верхнего уровня определяет, какие сообщения обрабатывать -->
 <xsl:template match="/">
   <xsl:choose>
       <xsl:when test="$message = '*'">
         <xsl:apply-templates select="*/Messages/*"/>
       </xsl:when>
       <xsl:when test="*/Messages/Message[Name=$message]">
         <xsl:apply-templates select="*/Messages/Message[Name=$message]"/>
       </xsl:when>
       <xsl:otherwise>
          <xsl:message terminate="yes">No such message name
          [<xsl:value-of select="$message"/>]</xsl:message>
       </xsl:otherwise>
    </xsl:choose>
 </xsl:template>

<!-- Если тип данных, представленных сообщением, хранится в репозитории, -->
```

```
<!-- сгенерировать заголовочный и исходный файл для его обертки -->
<xsl:template match="Message">
  <xsl:choose>
    <xsl:when test="key('dataTypes',DataTypeName)">
      <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                           mode="source">
        <xsl:with-param name="msg" select="Name"/>
      </xsl:apply-templates>
    </xsl:when>
    <xsl:otherwise>
        <xsl:message>Message name [<xsl:value-of select="Name"/>] uses data
        [<xsl:value-of select="DataTypeName"/>] that is not defined in the
         repository.</xsl:message>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Исходные файлы генерируются только, если тип данных в сообщении -
Structure. Возможен также тип данных XML. Если полезная нагрузка
представлена в формате XML, обертка не  генерируется. -->
<xsl:template match="Structure" mode="source">
  <xsl:param name="msg"/>

  <xsl:document href="{concat($generationDir,Name,'CGI',$sourceExt)}">

<xsl:text>
#include &lt;stdio.h&gt;
#include "cgi.h"
#include "</xsl:text>
<xsl:value-of select="Name"/><xsl:value-of select="$headerExt"/>
<xsl:text>"

void cgi_main(cgi_info *cgi)
{
    </xsl:text>
    <xsl:value-of select="Name"/>
    <xsl:text> data ;
    form_entry* form_data = get_form_entries(cgi) ;
</xsl:text>

  <xsl:for-each select="Members/Member">
    <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                         mode="variables">
```

```
        <xsl:with-param name="field" select="concat($msg,'_',Name)"/>
        <xsl:with-param name="var" select="Name"/>
      </xsl:apply-templates>
    </xsl:for-each>

  <xsl:for-each select="Members/Member">
    <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                         mode="load">
      <xsl:with-param name="field" select="concat('data.',Name)"/>
      <xsl:with-param name="var" select="Name"/>
    </xsl:apply-templates>
  </xsl:for-each>

<xsl:text>
&INDENT;//Поместить данные в очередь к тестируемому процессу
&INDENT;enqueData(data) ;

}</xsl:text>
</xsl:document>
</xsl:template>

<!-- Объявляем и инициализируем переменные для каждого поля -->
<xsl:template match="Structure" mode="variables">
  <xsl:param name="field"/>
  <xsl:param name="var"/>
    <xsl:for-each select="Members/Member">
      <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                mode="variables">
          <xsl:with-param name="field" select="concat($field,'_',Name)"/>
          <xsl:with-param name="var" select="$var"/>
       </xsl:apply-templates>
    </xsl:for-each>
</xsl:template>

<xsl:template match="*" mode="variables">
  <xsl:param name="field"/>
  <xsl:param name="var"/>

  <xsl:text>&INDENT;const char * </xsl:text>
  <xsl:value-of select="$var"/>
  <xsl:text> = parmval(form_data, "</xsl:text>
  <xsl:value-of select="$field"/>
  <xsl:text>");&#xa;</xsl:text>
</xsl:template>
```

```
<!-- Инициализируем данные для преобразованного значения -->
<xsl:template match="Structure" mode="load">
  <xsl:param name="field"/>
  <xsl:param name="var"/>
  <xsl:for-each select="Members/Member">
     <xsl:apply-templates select="key('dataTypes',DataTypeName)"
                          mode="load">
        <xsl:with-param name="field" select="concat($field,'.',Name)"/>
        <xsl:with-param name="var" select="concat($field,'_',Name)"/>
     </xsl:apply-templates>
  </xsl:for-each>
</xsl:template>

<xsl:template match="Primitive" mode="load">
  <xsl:param name="field"/>
  <xsl:param name="var"/>

  <xsl:text>&INDENT;</xsl:text>
  <xsl:value-of select="$field"/>
  <xsl:text> = </xsl:text>
  <xsl:value-of select="Name"/>
  <xsl:text>(</xsl:text>
  <xsl:value-of select="$var"/>
  <xsl:text>);&#xa;</xsl:text>
</xsl:template>

<xsl:template match="Enumeration" mode="load">
  <xsl:param name="field"/>
  <xsl:param name="var"/>

  <xsl:text>&INDENT;</xsl:text>
  <xsl:value-of select="$field"/>
  <xsl:text> = Enum</xsl:text>
  <xsl:value-of select="Name"/>
  <xsl:text>NameToVal(</xsl:text>
  <xsl:value-of select="$var"/>
  <xsl:text>");&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Если применить это преобразование к нашему репозиторию, будут сгенерированы CGI-программы для каждого сообщения. Например:

```
#include <stdio.h>
#include "cgi.h"
#include "msg_ids.h"
#include "AddStockOrderData.h"

void cgi_main(cgi_info *cgi)
{
      AddStockOrderData data ;
      form_entry* form_data = get_form_entries(cgi) ;
      const char * symbol = parmval(form_data, "ADD_STOCK_ORDER_symbol");
      const char * quantity = parmval(form_data, "ADD_STOCK_ORDER_quantity");
      const char * side = parmval(form_data, "ADD_STOCK_ORDER_side");
      const char * type = parmval(form_data, "ADD_STOCK_ORDER_type");
      const char * price = parmval(form_data, "ADD_STOCK_ORDER_price");
      data.symbol = StkSymbol(symbol);
      data.quantity = Shares(quantity);
      data.side = EnumBuyOrSellNameToVal(side);
      data.type = EnumOrderTypeNameToVal(type);
      data.price = Real(price);

    // Поместить данные в очередь к тестируемому процессу
      enqueData(ADD_STOCK_ORDER,data) ;

}
```

Обсуждение

В этом решении сделано несколько допущений. Во-первых, предполагается, что примитивные типы (например, `Shares`) – это классы, в которых имеется конструктор, преобразующий строку во внутреннее представление (например, `int`). Во-вторых, предполагается, что существуют функции-конверторы символических имен перечисляемых констант в их целочисленное представление. Как вы, наверное, догадались, такие функции легко сгенерировать по информации, хранящейся в репозитории, оставляем эту задачу в качестве упражнения. В-третьих, предполагается, что существует функция `enqueData`, которая знает, как посылать сообщение нужному процессу. Детали того, как эта функция взаимодействует с конкретным промежуточным слоем и находит подходящую очередь, зависят от реализации.

В общем, этот генератор еще следует «заточить» под собственные нужды. Однако, если в репозитории хранится достаточно информации, то автоматизировать можно многое. Без той автоматизации тестов, которая приведена в этом рецепте и в рецепте 12.6, программисту пришлось бы трудиться не покладая рук, особенно если состав и структура сообщений, обрабатываемых приложением, постоянно изменяются.

12.8. Генерация кода из UML-моделей, описанных на языке XMI

Задача

Требуется сгенерировать код по спецификациям на языке Unified Modeling Language (UML), но результат, который выдает генератор, встроенный в инструментальную программу для работы с UML, вам по каким-то причинам не нравится.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «СОСТОЯНИЕ»

Конечный автомат – полезный инструмент проектирования, позволяющий непосредственно моделировать программную систему, поведение который изменяется во времени по мере наступления различных событий. Часто конечные автоматы представляют в виде графов, вершины которых обозначают состояния, а ребра – переходы между состояниями в ответ на события.

Паттерн проектирования *Состояние* (State) описывает подход к построению конечного автомата, в котором конкретные состояния представляются классами, производными от абстрактного класса `State`. Отдельный класс `StateMachine` представляет конечный автомат в целом и содержит указатель на объект `State`, соответствующий текущему состоянию. Методы класса `StateMachine` делегируют текущему состоянию реализацию поведения в конкретном состоянии. При вызове любого из этих методов класс конкретного состояния может перевести владеющий им объект `StateMachine` в новое состояние.

Решение

XMI – это стандартизованное представление UML-моделей в формате XML, которое поддерживают многие инструменты моделирования (например, Rational Rose). Хотя официально цель XMI – обеспечить обмен информацией о модели между различными инструментами, этот язык можно использовать также и для генерации кода.

Большинство инструментов для работы с UML поддерживают генерацию кода, но редко идут дальше генерации скелета объектной модели. Так, все UML-генераторы, с которыми я знаком, принимают во внимание только информацию из диаграмм классов.

В примере ниже мы сгенерируем код, соответствующий паттерну «Состояние»[1], в котором будет использоваться информация из диаграммы классов (см. рис. 12.2) и диаграммы состояний (рис. 12.3).

Генератор кода предполагает, что проектировщик следовал таким соглашениям:

---

[1] Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес «Приемы объектно-ориентированного проектирования. Паттерны проектирования», ДМК-Пресс, Питер, 2007.

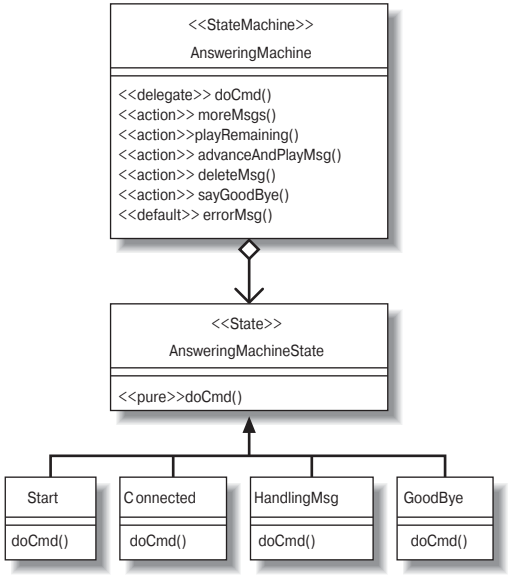


Рис. 12.2. Диаграмма классов, представляющих состояния

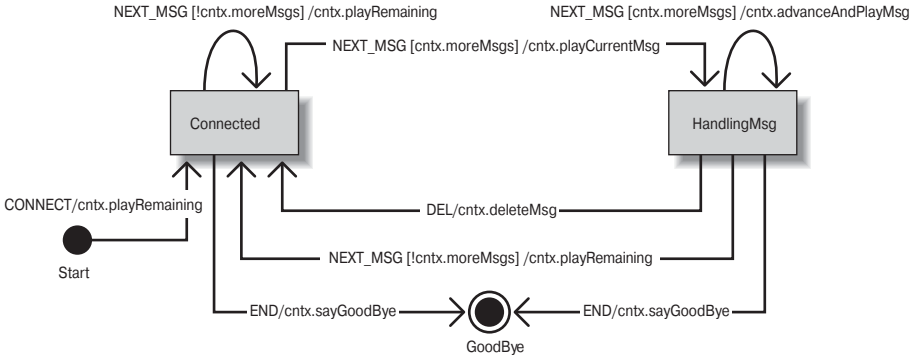


Рис. 12.3. Диаграмма состояний автомата, описывающего ответы на сообщения

1. Класс контекста состояния снабжен стереотипом `StateMachine`.
2. Абстрактный класс, которому наследуют классы конкретных состояний, снабжен стереотипом `State`.
3. Все операции контекста состояния, которые делегируют свою реализацию классам состояний, снабжены стереотипом `delegate`.
4. Все операции контекста состояния, используемые состояниями для реализации поведения конечного автомата, снабжены стереотипом `action`.
5. Единственное исключение из правила 4 составляет операция, которая должна вызываться, когда состояние не знает, что делать. Эта операция снабжена стереотипом `default` и обычно используется для обработки ошибок.

6. Имена состояний на UML-диаграмме состояний совпадают с именами конкретных классов, производных от интерфейса `State` (2).
7. *Действия* (actions) и *стражи* (guards), ассоциированные с переходами состояний, ссылаются на контекст и пользуются точными именами операций. *Действие* – это код, исполняемый в момент перехода, а *страж* – условие, которое должно удовлетворяться, чтобы данный переход был выбран.

В следующем примере показан конечный автомат, который реализует базовую функциональность, требуемую от процесса, обрабатывающего поступающие сообщения.

XMI-код, сгенерированный для этого простого примера, невозможно привести полностью, так как он огромен. В языке XMI применяются до нелепости длинные и трудновоспринимаемые имена. Мы приведем лишь небольшой фрагмент, описывающий класс `AnsweringMachineState`, чтобы вы могли составить представление о схеме. Но даже и его пришлось сократить.

```
<Foundation.Core.Class xmi.id="S.10011">
  <Foundation.Core.ModelElement.name>AnsweringMachineState
  </Foundation.Core.ModelElement.name>
  <Foundation.Core.ModelElement.stereotype>
    <Foundation.Extension_Mechanisms.Stereotype xmi.idref="G.22"/>
    <!-- state -->
  </Foundation.Core.ModelElement.stereotype>
  <Foundation.Core.GeneralizableElement.specialization>
    <Foundation.Core.Generalization xmi.idref="G.91"/>
    <!-- {Connected-&gt;AnsweringMachineState}{3D6782A402EE} -->
    <Foundation.Core.Generalization xmi.idref="G.92"/>
    <!-- {HandlingMsg-&gt;AnsweringMachineState}{3D6782EF0119} -->
    <Foundation.Core.Generalization xmi.idref="G.93"/>
    <!-- {Start-&gt;AnsweringMachineState}{3D67B2FD004E} -->
    <Foundation.Core.Generalization xmi.idref="G.94"/>
    <!-- {GoodBye-&gt;AnsweringMachineState}{3D67B31B02AF} -->
  </Foundation.Core.GeneralizableElement.specialization>
  <Foundation.Core.Classifier.associationEnd>
    <Foundation.Core.AssociationEnd xmi.idref="G.25"/>
  </Foundation.Core.Classifier.associationEnd>
  <Foundation.Core.Classifier.feature>
    <Foundation.Core.Operation xmi.id="S.10012">
      <Foundation.Core.ModelElement.name>doCmd
      </Foundation.Core.ModelElement.name>
      <Foundation.Core.ModelElement.visibility xmi.value="public"/>
      <Foundation.Core.Feature.ownerScope xmi.value="instance"/>
      <Foundation.Core.BehavioralFeature.isQuery xmi.value="false"/>
      <Foundation.Core.Operation.specification/>
      <Foundation.Core.Operation.isPolymorphic xmi.value="false"/>
      <Foundation.Core.Operation.concurrency xmi.value="sequential"/>
```

```
        <Foundation.Core.ModelElement.stereotype>
          <Foundation.Extension_Mechanisms.Stereotype xmi.idref="G.23"/>
          <!-- pure -->
        </Foundation.Core.ModelElement.stereotype>
        <Foundation.Core.BehavioralFeature.parameter>
          <Foundation.Core.Parameter xmi.id="G.76">

<Foundation.Core.ModelElement.name>cntx
</Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.visibility xmi.value="private"/>
            <Foundation.Core.Parameter.defaultValue>
              <Foundation.Data_Types.Expression>
                <Foundation.Data_Types.Expression.language/>
                <Foundation.Data_Types.Expression.body/>
              </Foundation.Data_Types.Expression>
            </Foundation.Core.Parameter.defaultValue>
            <Foundation.Core.Parameter.kind xmi.value="inout"/>
            <Foundation.Core.Parameter.type>
              <Foundation.Core.Class xmi.idref="S.10001"/>
              <!-- AnsweringMachine -->
            </Foundation.Core.Parameter.type>
          </Foundation.Core.Parameter>
          <Foundation.Core.Parameter xmi.id="G.77">

<Foundation.Core.ModelElement.name>cmd
</Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.visibility xmi.value="private"/>
            <Foundation.Core.Parameter.defaultValue>
              <Foundation.Data_Types.Expression>
                <Foundation.Data_Types.Expression.language/>
                <Foundation.Data_Types.Expression.body/>
              </Foundation.Data_Types.Expression>
            </Foundation.Core.Parameter.defaultValue>
            <Foundation.Core.Parameter.kind xmi.value="inout"/>
            <Foundation.Core.Parameter.type>
              <Foundation.Core.DataType xmi.idref="G.65"/>
              <!-- Command ->
            </Foundation.Core.Parameter.type>
          </Foundation.Core.Parameter>
          <Foundation.Core.Parameter xmi.id="G.78">
            <Foundation.Core.ModelElement.name>doCmd.Return
            </Foundation.Core.ModelElement.name>
            <Foundation.Core.ModelElement.visibility xmi.value="private"/>
            <Foundation.Core.Parameter.defaultValue>
```

```
            <Foundation.Data_Types.Expression>
              <Foundation.Data_Types.Expression.language/>
              <Foundation.Data_Types.Expression.body/>
            </Foundation.Data_Types.Expression>
          </Foundation.Core.Parameter.defaultValue>
          <Foundation.Core.Parameter.kind xmi.value="return"/>
          <Foundation.Core.Parameter.type>
            <Foundation.Core.DataType xmi.idref="G.67"/>
            <!-- void -->
          </Foundation.Core.Parameter.type>
        </Foundation.Core.Parameter>
      </Foundation.Core.BehavioralFeature.parameter>
    </Foundation.Core.Operation>
  </Foundation.Core.Classifier.feature>
</Foundation.Core.Class>
```

Показанная ниже таблица стилей (довольно длинная) преобразует этот XMI-код в реализацию конечного автомата на C++, устроенную в соответствии с паттерном «Состояние». Только от полного отчаяния я прибегнул к XML-компонентам, чтобы сократить длинные XMI-имена. Вообще говоря, я не рекомендую такую практику, но в данном случае это был единственный способ уместить XSLT-код на печатной странице. Заодно устраняется «шум», свойственный схеме XMI DTD. Обратите внимание на повсеместное применение ключей, объясняющееся тем, что в XMI активно используются перекрестные ссылки в форме атрибутов `xmi.id` и `xmi.idref`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
  <!--=================================================-->
  <!-- Конструкции XMI, применяемые на верхнем уровне  -->
  <!--=================================================-->
  <!ENTITY BE "Behavioral_Elements">
  <!ENTITY SM "&BE;.State_Machines">
  <!ENTITY SMC "&SM;.StateMachine.context">
  <!ENTITY SMT "&SM;.StateMachine.top">
  <!ENTITY MM "Model_Management.Model">
  <!ENTITY FC "Foundation.Core">
  <!ENTITY FX "Foundation.Extension_Mechanisms">

  <!--=================================================-->
  <!-- Сокращения для основных элементов XMI, представляющих -->
  <!-- наибольший интерес для данной таблицы стилей.   -->
  <!--=================================================-->
  <!--The model as a whole -->
  <!ENTITY MODEL "XMI/XMI.content/&MM;">
  <!--Some generic kind of UML element -->
```

```
  <!ENTITY ELEM "&FC;.Namespace.ownedElement">
  <!--Elements of state machines -->
  <!ENTITY STATEMACH "&MODEL;/&ELEM;/&SM;.StateMachine">
  <!ENTITY STATE "&SM;.CompositeState">
  <!ENTITY SUBSTATE "&STATE;.substate">
  <!ENTITY PSEUDOSTATE "&SM;.PseudoState">
  <!ENTITY PSEUDOSTATE2 "&SMT;/&STATE;/&SUBSTATE;/&PSEUDOSTATE;">
  <!ENTITY ACTION "&BE;.Common_Behavior.ActionSequence/&BE;.Common_Behavior.
ActionSequence.action">
  <!ENTITY GUARD "&SM;.Transition.guard/&SM;.Guard/&SM;.Guard.expression">
  <!--The association as a whole -->
  <!ENTITY ASSOC "&FC;.Association">
  <!--The connection part of the association-->
  <!ENTITY CONN "&ASSOC;.connection">
  <!--The ends of an association. -->
  <!ENTITY END "&ASSOC;End">
  <!ENTITY CONNEND "&CONN;/&END;">
  <!ENTITY ENDType "&END;.type">
  <!-- Класс UML -->
  <!ENTITY CLASS "&FC;.Class">
  <!--The name of some UML entity -->
  <!ENTITY NAME "&FC;.ModelElement.name">
  <!-- Операция -->
  <!ENTITY OP "&FC;.Operation">
  <!-- Параметр -->
  <!ENTITY PARAM "&FC;.Parameter">
  <!ENTITY PARAM2 "&FC;.BehavioralFeature.parameter/&PARAM;">
  <!-- Тип данных параметра -->
  <!ENTITY PARAMTYPE "&PARAM;.type/&FC;.DataType">
  <!-- Стереотип UML -->
  <!ENTITY STEREOTYPE "&FC;.ModelElement.stereotype/&FX;.Stereotype">
  <!-- Отношение наследования -->
  <!ENTITY SUPERTYPE "&FC;.Generalization.supertype">
  <!ENTITY GENERALIZATION "&FC;.GeneralizableElement. generalization/&FC;.
Generalization">
  <!--================================================-->
  <!-- Форматирование                                 -->
  <!--================================================-->

  <!-- Служит для управления отступами -->
  <!ENTITY INDENT "    ">
  <!ENTITY INDENT2 "&INDENT;&INDENT;">
]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
  <xsl:output method="text"/>

  <!-- Индексировать классы по идентификатору id -->
  <xsl:key name="classKey" match="&CLASS;" use="@xmi.id"/>
  <xsl:key name="classKey" match="&CLASS;" use="&NAME;"/>

  <!-- Индексировать стереотипы по имени и атрибуту xmi.id -->
  <xsl:key name="stereotypeKey" match="&FX;.Stereotype" use="@xmi.id"/>
  <xsl:key name="stereotypeKey" match="&FX;.Stereotype" use="&NAME;"/>

  <!-- Индексировать типы данных по id -->
  <xsl:key name="dataTypeKey" match="&FC;.DataType" use="@xmi.id"/>

  <!-- Индексировать ассоциации по оконечным классам -->
  <xsl:key name="associationKey" match="&ASSOC;" use="&CONNEND;/&ENDType;/&FC;.
Interface/@xmi.idref"/>
  <xsl:key name="associationKey" match="&ASSOC;" use="&CONNEND;/&ENDType;/
&CLASS;/@xmi.idref"/>

  <!-- Индексировать обобщения по id -->
  <xsl:key name="generalizationKey" match="&FC;.Generalization"
use="@xmi.id"/>

  <!-- Индексировать состояния и псевдосостояния одним индексом по id -->
  <xsl:key name="stateKey" match="&SM;.SimpleState" use="@xmi.id"/>
  <xsl:key name="stateKey" match="&PSEUDOSTATE;" use="@xmi.id"/>

  <!-- Индексировать переходы состояний по id -->
  <xsl:key name="transKey" match="&SM;.Transition" use="@xmi.id"/>

  <!-- Индексировать события перехода по id -->
  <xsl:key name="eventsKey" match="&SM;.SignalEvent" use="@xmi.id"/>

  <!-- XMI-идентификатор стереотипов, применяемых для кодирования -->
  <!-- паттерна Состояние на языке UML -->
  <xsl:variable name="STATE_MACH"
                select="key('stereotypeKey','StateMachine')/@xmi.id"/>
  <xsl:variable name="STATE"
                select="key('stereotypeKey','state')/@xmi.id"/>
  <xsl:variable name="DELEGATE"
                select="key('stereotypeKey','delegate')/@xmi.id"/>
  <xsl:variable name="ACTION"
                select="key('stereotypeKey','action')/@xmi.id"/>
  <xsl:variable name="DEFAULT"
                select="key('stereotypeKey','default')/@xmi.id"/>
```

```
<xsl:variable name="PURE"
              select="key('stereotypeKey','pure')/@xmi.id"/>

<!-- Для генерации исходного файла за пять шагов используются режимы -->
<xsl:template match="/">

  <!-- Объявляем интерфейс состояния -->
  <xsl:apply-templates mode="stateInterfaceDecl"
        select="&MODEL;/&ELEM;/&CLASS;[&STEREOTYPE;/@xmi.idref =
                                                    $STATE_MACH]"/>

  <!-- Объявляем конкретные состояния -->
  <xsl:apply-templates mode="concreteStatesDecl"
        select="&MODEL;/&ELEM;/&CLASS;[not(&STEREOTYPE;)]"/>

  <!-- Объявляем класс контекста состояния -->
  <xsl:apply-templates mode="stateDecl"
        select="&MODEL;/&ELEM;/&CLASS;[&STEREOTYPE;/@xmi.idref =
                                                    $STATE_MACH]"/>

  <!-- Реализуем состояния -->
  <xsl:apply-templates mode="concreteStatesImpl"
        select="&MODEL;/&ELEM;"/>

  <!-- Реализуем контекст -->
  <xsl:apply-templates mode="stateContextImpl"
        select="&MODEL;/&ELEM;/&CLASS;[&STEREOTYPE;/@xmi.idref =
                                    $STATE_MACH]"/>

</xsl:template>

<!-- ОБЪЯВЛЕНИЕ КОНТЕКСТА СОСТОЯНИЯ -->
<xsl:template match="&CLASS;" mode="stateDecl">

 <!-- Найти класс, ассоциированный с контекстом этого состояния, -->
<!-- в котором реализовано состояние. Это будет тип переменной-члена, -->
<!-- указывающей на текущее состояние конечного автомата -->
 <xsl:variable name="stateImplClass">
    <xsl:variable name="stateClassId" select="@xmi.id"/>
    <xsl:for-each select="key('associationKey',$stateClassId)">
      <xsl:variable name="assocClassId"
                    select="&CONNEND;[&ENDType;/*/@xmi.idref !=
                                $stateClassId]/&ENDType;/*/@xmi.idref"/>
      <xsl:if test="key('classKey',$assocClassId)/&STEREOTYPE;/@xmi.idref
                                                        = $STATE">
```

```
        <xsl:value-of select="key('classKey',$assocClassId)/&NAME;"/>
      </xsl:if>
    </xsl:for-each>
  </xsl:variable>

  <xsl:variable name="className" select="&NAME;"/>

  <xsl:text>&#xa;class </xsl:text>
  <xsl:value-of select="$className"/>
  <xsl:text>&#xa;{&#xa;public:&#xa;&#xa;</xsl:text>

  <!-- Объявление конструктора -->
  <xsl:text>&INDENT;</xsl:text>
 <xsl:value-of select="$className"/>::<xsl:value-of select="$className"/>
  <xsl:text>();&#xa;&#xa;</xsl:text>

 <!-- Снабженные стереотипом delegate операции делегируют выполнение -->
 <!-- текущему состоянию -->
  <xsl:apply-templates
                select="*/&OP;[&STEREOTYPE;/@xmi.idref = $DELEGATE]"
                mode="declare"/>

  <!-- void changeState(AbstractState& newState) -->
  <xsl:text>&INDENT;void changeState(</xsl:text>
  <xsl:value-of select="$stateImplClass"/>
  <xsl:text>&amp; newSate) ;&#xa;</xsl:text>

  <xsl:text>&#xa;&#xa;</xsl:text>
 <!-- Операции, не снабженные стереотипом delegate, — это все прочие -->
  <!-- операции контекста, которые может вызывать состояние -->
  <xsl:apply-templates
           select="*/&OP;[&STEREOTYPE;/@xmi.idref != $DELEGATE]"
           mode="declare"/>
  <xsl:text>&#xa;private:&#xa;&#xa;</xsl:text>
  <xsl:text>&INDENT;</xsl:text>
  <xsl:value-of select="$stateImplClass"/>
  <xsl:text>* m_State ;</xsl:text>
  <xsl:text>&#xa;&#xa;} ;&#xa;&#xa;</xsl:text>

</xsl:template>

<!-- ОБЪЯВЛЕНИЕ КОНКРЕТНЫХ СОСТОЯНИЙ -->

<xsl:template match="&CLASS;" mode="concreteStatesDecl">
```

```
      <xsl:text>class </xsl:text>
    <xsl:value-of select="&NAME;"/>
      <xsl:call-template name="baseClass"/>
      <xsl:text>&#xa;{&#xa;public:&#xa;&#xa;</xsl:text>
      <!-- Concrete States are Singletons so we generate an -->
      <!-- instance method -->
      <xsl:text>&INDENT;static </xsl:text>
      <xsl:value-of select="&NAME;"/>
      <xsl:text>&amp; instance() ;</xsl:text>
      <xsl:text>&#xa;&#xa;private:&#xa;&#xa;</xsl:text>
      <!-- Конструкторы синглетных классов должны быть защищенными -->
      <xsl:text>&INDENT;</xsl:text>
      <xsl:value-of select="&NAME;"/>
      <xsl:text>() {} &#xa;</xsl:text>
      <xsl:text>&INDENT;</xsl:text>
      <xsl:value-of select="&NAME;"/>(const <xsl:value-of select="&NAME;"/>
      <xsl:text>&amp;) {} &#xa;</xsl:text>
      <xsl:text>&INDENT;void operator =(const </xsl:text>
    <xsl:value-of select="&NAME;"/>
    <xsl:text>&amp;) {} &#xa;</xsl:text>
    <xsl:apply-templates select="*/&OP;" mode="declare"/>
    <xsl:text>
} ;

</xsl:text>
  </xsl:template>

  <!-- Шаблоны, используемые для объявления классов, содержащих -->
  <!-- только открытые члены -->

  <xsl:template match="&CLASS;" mode="declare">
    <xsl:text>class&#x20;</xsl:text>
    <xsl:value-of select="&NAME;"/>
    <xsl:text>&#xa;{&#xa;public:&#xa;&#xa;</xsl:text>
    <xsl:apply-templates select="*/&OP;" mode="declare"/>
    <xsl:text>&#xa;} ;&#xa;</xsl:text>
  </xsl:template>

  <xsl:template match="&OP;" mode="declare">
    <xsl:variable name="returnTypeId"
                          select="&PARAM2;[&PARAM;.kind/@xmi.value =
                                  'return']/&PARAMTYPE;/@xmi.idref"/>
      <xsl:text>&INDENT;</xsl:text>
    <xsl:if test="&STEREOTYPE;/@xmi.idref = $PURE">
```

```
      <xsl:text>virtual </xsl:text>
   </xsl:if>
   <xsl:value-of select="key('dataTypeKey',$returnTypeId)/&NAME;"/>
      <xsl:text>&#x20;</xsl:text>
   <xsl:value-of select="&NAME;"/>
    <xsl:text>(</xsl:text>
    <xsl:call-template name="parameters"/>
    <xsl:text>)</xsl:text>
    <xsl:if test="&STEREOTYPE;/@xmi.idref = $PURE">
      <xsl:text> = 0 </xsl:text>
    </xsl:if>
    <xsl:text>;&#xa;</xsl:text>
 </xsl:template>

<!-- Игнорировать лишние текстовые узлы -->
<xsl:template match="text()" mode="declare"/>

<!-- ОБЪЯВЛЕНИЕ ИНТЕРФЕЙСА СОСТОЯНИЯ -->
  <xsl:template match="&CLASS;" mode="stateInterfaceDecl">

    <xsl:text>//Опережающие объявления&#xa;</xsl:text>
    <xsl:text>class </xsl:text>
    <xsl:value-of select="&NAME;"/>
    <xsl:text>;&#xa;&#xa;</xsl:text>

    <xsl:variable name="stateClassId" select="@xmi.id"/>
    <!-- Найти класс, ассоциированный с контекстом состояния -->
    <xsl:for-each select="key('associationKey',$stateClassId)">
      <xsl:variable name="assocClassId"
                  select="&CONNEND;[&ENDType;/*/@xmi.idref !=
                                  $stateClassId]/&ENDType;/*/@xmi.idref"/>
       <xsl:if test="key('classKey',$assocClassId)/&STEREOTYPE;/@xmi.idref =
                                                                  $STATE">
         <xsl:apply-templates select="key('classKey',$assocClassId)"
                             mode="declare"/>
      </xsl:if>
    </xsl:for-each>
    <xsl:text>&#xa;&#xa;</xsl:text>
  </xsl:template>

  <!-- Игнорировать лишние текстовые узлы -->
  <xsl:template match="text()" mode="stateInterfaceDecl"/>

  <!-- РЕАЛИЗАЦИЯ КОНТЕКСТА СОСТОЯНИЯ -->
```

```
    <xsl:template match="&CLASS;" mode="stateContextImpl">

      <xsl:variable name="stateImplClass">
        <xsl:variable name="stateClassId" select="@xmi.id"/>
        <xsl:for-each select="key('associationKey',$stateClassId)">
          <xsl:variable name="assocClassId"
               select="&CONNEND;[&ENDType;/*/@xmi.idref !=
                                $stateClassId]/&ENDType;/*/@xmi.idref"/>
          <xsl:if test="key('classKey',$assocClassId)/&STEREOTYPE;/@xmi.idref
                                                              = $STATE">
            <xsl:value-of select="key('classKey',$assocClassId)/&NAME;"/>
          </xsl:if>
        </xsl:for-each>
      </xsl:variable>

      <xsl:variable name="className" select="&NAME;"/>
      <xsl:text>//Конструктор&#xa;</xsl:text>
      <xsl:value-of select="$className"/>::<xsl:value-of
                        select="$className"/>
      <xsl:text>()&#xa;</xsl:text>
      <xsl:text>{&#xa;</xsl:text>
      <xsl:text>&INDENT;//Инициализировать конечный автомат в начальном
состоянии &#xa;</xsl:text>
      <xsl:variable name="startStateName">
        <xsl:call-template name="getStartState">
          <xsl:with-param name="classId" select="@xmi.id"/>
        </xsl:call-template>
      </xsl:variable>
      <xsl:text>&INDENT;m_State = &amp;</xsl:text>
      <xsl:value-of select="$startStateName"/>
      <xsl:text>::instance() ;&#xa;</xsl:text>
      <xsl:text>}&#xa;&#xa;</xsl:text>

      <!-- void changeState(AbstractState& newState) -->
      <xsl:text>void </xsl:text>
      <xsl:value-of select="$className"/>
      <xsl:text>::changeState(</xsl:text>
      <xsl:value-of select="$stateImplClass"/>
      <xsl:text>&amp; newState)&#xa;</xsl:text>
      <xsl:text>{&#xa;</xsl:text>
      <xsl:text>&INDENT;m_State = &amp;newState;</xsl:text>
      <xsl:text>&#xa;}&#xa;&#xa;</xsl:text>

      <xsl:for-each select="*/&OP;[&STEREOTYPE;/@xmi.idref = $DELEGATE]">
        <xsl:variable name="returnTypeId"
```

```
                select="&PARAM2;[&PARAM;.kind/@xmi.value =
                                'return']/&PARAMTYPE;/@xmi.idref"/>
    <xsl:value-of select="key('dataTypeKey',$returnTypeId)/&NAME;"/>
    <xsl:text>&#x20;</xsl:text>
    <xsl:value-of select="$className"/>::<xsl:value-of select="&NAME;"/>
    <xsl:text>(</xsl:text>
    <xsl:call-template name="parameters"/>
    <xsl:text>)&#xa;</xsl:text>
    <xsl:text>{&#xa;</xsl:text>
    <xsl:text>&INDENT;m_State-></xsl:text>
   <xsl:value-of select="&NAME;"/>
    <xsl:text>(*this, </xsl:text>
   <xsl:for-each select="&PARAM2;[&PARAM;.kind/@xmi.value != 'return']">
      <xsl:value-of select="&NAME;"/>
      <xsl:if test="position() != last()">
        <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:for-each>
    <xsl:text>);&#xa;</xsl:text>
    <xsl:text>}&#xa;&#xa;</xsl:text>
  </xsl:for-each>

  <xsl:for-each select="*/&OP;[&STEREOTYPE;/@xmi.idref != $DELEGATE]">
    <xsl:variable name="returnTypeId"
                select="&PARAM2;[&PARAM;.kind/@xmi.value =
                                'return']/&PARAMTYPE;/@xmi.idref"/>
    <xsl:value-of select="key('dataTypeKey',$returnTypeId)/&NAME;"/>
    <xsl:text>&#x20;</xsl:text>
    <xsl:value-of select="$className"/>::<xsl:value-of select="&NAME;"/>
    <xsl:text>(</xsl:text>
    <xsl:call-template name="parameters"/>
    <xsl:text>)&#xa;</xsl:text>
    <xsl:text>{&#xa;</xsl:text>
    <xsl:text>&INDENT;//!TODO: Реализовать это действие&#xa;</xsl:text>
    <xsl:text>}&#xa;&#xa;</xsl:text>
  </xsl:for-each>
</xsl:template>
<xsl:template match="text()" mode="stateContextImpl"/>

<!-- РЕАЛИЗАЦИЯ КОНКРЕТНЫХ СОСТОЯНИЙ -->
<xsl:template match="&CLASS;" mode="concreteStatesImpl">
  <xsl:variable name="classId" select="@xmi.id"/>
  <xsl:variable name="stateMachine"
                select="/&STATEMACH;[&SMC;/&CLASS;/@xmi.idref =
```

```
                                                                $classId]"/>
    <!-- Для каждого состояния автомата сгенерировать его реализацию -->
    <xsl:for-each select="$stateMachine//&PSEUDOSTATE; |
                                        $stateMachine//&SM;.SimpleState">
      <xsl:call-template name="generateState">
        <xsl:with-param name="stateClass" select="key('classKey',&NAME;)"/>
      </xsl:call-template>
    </xsl:for-each>
  </xsl:template>

  <!-- Игнорировать лишние текстовые узлы -->
  <xsl:template match="text()" mode="concreteStatesImpl"/>

  <xsl:template name="generateState">
    <!-- Это XMI-класс, соответствующий данному состоянию -->
    <xsl:param name="stateClass"/>
    <!-- Текущий контекст — это некоторое состояние -->
    <xsl:variable name="state" select="."/>
    <xsl:variable name="className" select="&NAME;"/>
    <xsl:if test="$className != $stateClass/&NAME;">
      <xsl:message terminate="yes">Нет соответствия между состоянием
и классом!</xsl:message>
    </xsl:if>
    <xsl:for-each select="$stateClass/*/&OP;">
      <xsl:variable name="returnTypeId"
                    select="&PARAM2;[&PARAM;.kind/@xmi.value =
                                        'return']/&PARAMTYPE;/@xmi.idref"/>
      <xsl:value-of select="key('dataTypeKey',$returnTypeId)/&NAME;"/>
      <xsl:text>&#x20;</xsl:text>
      <xsl:value-of select="$className"/>::<xsl:value-of select="&NAME;"/>
      <xsl:text>(</xsl:text>
      <xsl:call-template name="parameters"/>
      <xsl:text>)&#xa;</xsl:text>
      <xsl:text>{&#xa;</xsl:text>
      <xsl:for-each
           select="$state/&SM;.StateVertex.outgoing/&SM;.Transition">
        <xsl:call-template name="generateStateBody">
          <xsl:with-param name="transition"
           select="key('transKey',@xmi.idref)"/>
        </xsl:call-template>
      </xsl:for-each>
      <xsl:text>&INDENT2;context.errorMsg() ;&#xa;</xsl:text>
      <xsl:text>}&#xa;&#xa;</xsl:text>
    </xsl:for-each>
```

```
  </xsl:template>

  <xsl:template name="generateStateBody">
    <xsl:param name="transition"/>
    <xsl:text>&INDENT;if (cmd == </xsl:text>
    <xsl:variable name="eventId"

select="$transition/&SM;.Transition.trigger/&SM;.SignalEvent/@xmi.idref"/>
    <xsl:value-of select="key('eventsKey',$eventId)/&NAME;"/>
    <xsl:if test="$transition/&SM;.Transition.guard">
      <xsl:text> &amp;&amp; </xsl:text>
      <xsl:value-of
     select="$transition/&GUARD;/*/Foundation.Data_Types.Expression.body"/>
      <xsl:text>()</xsl:text>
    </xsl:if>
    <xsl:text>)&#xa;</xsl:text>
    <xsl:text>&INDENT;{&#xa;</xsl:text>
    <xsl:text>&INDENT2;</xsl:text>
    <xsl:value-of
         select="$transition/&SM;.Transition.effect/&ACTION;/*/&NAME;"/>
    <xsl:text>() ;&#xa;</xsl:text>
    <xsl:variable name="targetStateId"
                 select="$transition/&SM;.Transition.target/*/@xmi.idref"/>
    <xsl:if test="$targetStateId != $transition/@xmi.id"/>
      <xsl:text>&INDENT2;cntx.changeState(</xsl:text>
      <xsl:value-of select="key('stateKey',$targetStateId)/&NAME;"/>
      <xsl:text>::instance());&#xa;</xsl:text>
    <xsl:text>&INDENT;}&#xa;</xsl:text>
    <xsl:text>&INDENT;else&#xa;</xsl:text>
  </xsl:template>

  <!-- Генерируем параметры функций -->
  <xsl:template name="parameters">
    <xsl:for-each select="&PARAM2;[&PARAM;.kind/@xmi.value != 'return']">
      <xsl:choose>
        <xsl:when test="&PARAMTYPE;">
          <xsl:value-of
                select="key('dataTypeKey',&PARAMTYPE;/@xmi.idref)/&NAME;"/>
        </xsl:when>
        <xsl:when test="&PARAM;.type/&CLASS;">
          <xsl:value-of
            select="key('classKey',
                        &PARAM;.type/&CLASS;/@xmi.idref)/&NAME;"/>
          <xsl:text>&amp;</xsl:text>
```

```
        </xsl:when>
      </xsl:choose>
      <xsl:text>&#x20;</xsl:text>
      <xsl:value-of select="&NAME;"/>
      <xsl:if test="position() != last()">
        <xsl:text>, </xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:template>

  <!-- Генерируем базовые классы -->
  <xsl:template name="baseClass">
    <xsl:if test="&GENERALIZATION;">
      <xsl:text> : </xsl:text>
      <xsl:for-each select="&GENERALIZATION;">
        <xsl:variable name="genAssoc"
                      select="key('generalizationKey',@xmi.idref)"/>
        <xsl:text>public </xsl:text>
        <xsl:value-of
   select="key('classKey',
                $genAssoc/&SUPERTYPE;/&CLASS;/@xmi.idref)/&NAME;"/>
        <xsl:if test="position() != last()">
          <xsl:text>, </xsl:text>
        </xsl:if>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>
  <xsl:template name="getStartState">
    <xsl:param name="classId"/>
    <xsl:variable name="stateMachine"
                  select="/&STATEMACH;[&SMC;/&CLASS;/@xmi.idref =
                                                        $classId]"/>
    <xsl:value-of
                select="$stateMachine/&PSEUDOSTATE2;
                                      [&PSEUDOSTATE;.kind/@xmi.value =
'initial']/&NAME;"/>
  </xsl:template>
</xsl:stylesheet>
```

Вот фрагмент сгенерированного кода на C++:

```
//Опережающие объявления
class AnsweringMachine;

class AnsweringMachineState
```

```
{
public:

      virtual void doCmd(AnsweringMachine& cntx, Command cmd) = 0 ;

} ;

class Connected : public AnsweringMachineState
{
public:

      static Connected& instance(  ) ;

private:

      Connected(  ) {  }
      Connected(const Connected&) {  }
      void operator =(const Connected&) {  }
      void doCmd(AnsweringMachine& cntx, Command cmd);

} ;

class HandlingMsg : public AnsweringMachineState
{
public:

    static HandlingMsg& instance(  ) ;

private:

      HandlingMsg(  ) {  }
      HandlingMsg(const HandlingMsg&) {  }
      void operator =(const HandlingMsg&) {  }
      void doCmd(AnsweringMachine& cntx, Command cmd);

} ;

//!ПРОЧИЕ СОСТОЯНИЯ ОПУЩЕНЫ ДЛЯ ЭКОНОМИИ МЕСТА...

class AnsweringMachine
{
public:

    AnsweringMachine::AnsweringMachine(  );
```

```
    void doCmd(Command cmd);
    void changeState(AnsweringMachineState& newSate) ;

    bool moreMsgs(  );
    void playRemaining(  );
    void playCurrentMsg(  );
    void advanceAndPlayMsg(  );
    void deleteMsg(  );
    void sayGoodBye(  );
    void errorMsg(  );

private:

    AnsweringMachineState* m_State ;

} ;

void Connected::doCmd(AnsweringMachine& cntx, Command cmd)
{
  if (cmd =  = NEXT_MSG && cntx.moreMsgs(  ))
  {
      cntx.playCurrentMsg(  ) ;
      cntx.changeState(HandlingMsg::instance(  ));
  }
  else
  if (cmd =  = NEXT_MSG && !cntx.moreMsgs(  ))
  {
      cntx.playRemaining(  ) ;
      cntx.changeState(Connected::instance(  ));
  }
  else
  if (cmd =  = END)
  {
      cntx.sayGoodBye(  ) ;
      cntx.changeState(GoodBye::instance(  ));
  }
  else
      context.errorMsg(  ) ;
}

//!ПРОЧИЕ СОСТОЯНИЯ ОПУЩЕНЫ ДЛЯ ЭКОНОМИИ МЕСТА...

//Конструктор
AnsweringMachine::AnsweringMachine(  )
```

```
{
    //Инициализировать конечный автомат в начальном состоянии
    m_State = &Start::instance(  ) ;
}

void AnsweringMachine::changeState(AnsweringMachineState& newState)
{
    m_State = &newState;
}

void AnsweringMachine::doCmd(Command cmd)
{
    m_State->doCmd(*this, cmd);
}

bool AnsweringMachine::moreMsgs(  )
{
    //!TODO: Реализовать это действие
}

void AnsweringMachine::playRemaining(  )
{
    //!TODO: Реализовать это действие
}

void AnsweringMachine::playCurrentMsg(  )
{
    //!TODO: Реализовать это действие
}

//!ПРОЧИЕ ДЕЙСТВИЯ ОПУЩЕНЫ ДЛЯ ЭКОНОМИИ МЕСТА...
```

Обсуждение

Желание написать собственный генератор код по UML-модели может возникнуть в трех случаях. Во-первых, вам не нравится стиль или сам код, созданный штатным кодогенератором, входящим в состав инструментальной программы. Во-вторых, инструмент не умеет генерировать код для нужного вам языка. Например, я не знаю инструмента, который генерировал бы код на языке Python[1]. В-третьих, вы хотите автоматизировать генерацию кода для какого-то

[1] Не хочу сказать, что я большой поклонник языка, в котором нет фигурных скобок, но у него есть немало приверженцев.

высокоуровневого паттерна или программной службы, часто встречающейся в приложении. Именно последняя причина натолкнула нас на идею, реализованную в этом примере.

Создание универсального механизма для генерации кода на различных языках, построенного на базе XMI и XSLT, – это интересный проект, но он выходит далеко за пределы одного примера. Приведенное решение – не окончательный генератор промышленного качества, а лишь подступы к нему. В частности, можно было бы реализовать четыре важных усовершенствования.

Во-первых, таблицу стилей следует разбить на небольшие компоненты, которые генерируют отдельные конструкции на целевом языке. Примерно так мы поступили в главе 9, когда генерировали SVG-код.

Во-вторых, таблица стилей должна извлекать из UML-модели больше информации, в частности о контроле доступа, в не «зашивать» в код произвольные допущения.

В-третьих, в генераторе следует применить расширения XSLT 1.0 или XSLT 2.0, чтобы разбить код на несколько заголовочных и исходных файлов, а не помещать все в один монолитный исходный файл.

В-четвертых, хотя это и не так важно, можно было бы поддержать несколько стилей оформления кода. Например, я генерировал код на C++ в стиле Олмана (Allman), хотя многие (по недоразумению?) предпочитают стиль Кернигана и Ричи (K&R) (*http://www.tuxedo.org/~esr/jargon/html/entry/indent-style.html*). Но можно было бы просто пропустить результирующий код через какой-нибудь автономный форматер и вообще не задумываться о форматировании.

12.9. Генерация XSLT из XSLT

Задача

Требуется сгенерировать XSLT-код из другого XML-представления. Или преобразовать один код на XSLT или псевдо-XSLT в другой.

Решение

Иногда меня раздражают два аспекта управляющих конструкций в XSLT. Первый – отсутствие конструкции `if-then-elsif-else`; второй – отсутствие настоящих циклов. Конечно, я знаю о существовании команд `xsl:choose` и `xsl:for-each`, но у каждой из них есть недостатки. Так, `xsl:choose` мне не нравится, потому что у элемента `choose` нет никакого иного назначения, кроме организации дополнительного уровня вложенности. А `xsl:for-each` – не столько цикл, сколько конструкция для итерирования. Чтобы эмулировать поведение цикла со счетчиком, приходится прибегать к рекурсии или методу Пиза (см. рецепт 2.5), а это не красиво.

В следующем примере иллюстрируется преобразование в XSLT кода, написанного на псевдо-XSLT, в котором имеются элементы `xslx:elsif`, `xslx:else` и `xslx:loop`. Поскольку на самом деле их нет, то мы создадим таблицу стилей, которая заменит эти элементы реально существующими конструкциями. Одновременное

наличие элементов `xsl:if` и `xslx:if` выглядит нелепо, но было бы неправильно использовать стандартное пространство имен XSLT для дополнительных элементов; не исключено, что в будущей версии стандарта XSLT они будут определены.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xslx="http://www.ora.com/XSLTCookbook/ExtendedXSLT" >

<xsl:output method="text"/>

<xsl:template match="foo">
  <xslx:if test="bar">
     <xsl:text>Нередко рядом с bar встречается и foo!</xsl:text>
  </xslx:if>
  <xslx:elsif test="baz">
     <xsl:text>Слово baz - верный признак сленга</xsl:text>
  </xslx:elsif>
  <xslx:else>
     <xslx:loop param="i" init="0" test="$i &lt; 10" incr="1">
        <xsl:text>Гм, сказать-то нечего, но повторю это 10 раз.</xsl:text>
     </xslx:loop>
  </xslx:else>
  <xslx:loop param="i" init="10" test="$i >= 0" incr="-1">
     <xslx:loop param="j" init="10" test="$j >= 0" incr="-1">
        <xsl:text>&#xa;</xsl:text>
        <xsl:value-of select="$i * $j"/>
     </xslx:loop>
  </xslx:loop>
  <xslx:if test="foo">
        <xsl:text>foo foo! Никто не говорит foo foo!</xsl:text>
  </xslx:if>
  <xslx:else>
     <xsl:text>Ну и ладно!</xsl:text>
  </xslx:else>
</xsl:template>

</xsl:stylesheet>
```

А ниже приведена таблица стилей, которая генерирует настоящий XSLT-код из такого псевдокода. Для простоты мы не включили некоторые семантические проверки, например, наличие нескольких `xsl:else` при единственном `xsl:if` или дублирование параметров во вложенных циклах:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xslx="http://www.ora.com/XSLTCookbook/ExtendedXSLT"
```

```
   xmlns:xso="dummy" >

<!-- Повторно используем тождественное преобразование -->
<!-- для копирования корректного XSLT-кода -->
<xsl:import href="../util/copy.xslt"/>

<!-- НЕ позволяем процессору выполнять форматирование с -->
<!-- помощью indent = yes, так как это испортило бы узлы xsl:text -->
<xsl:output method="xml" version="1.0" encoding="UTF-8" />

<!-- Когда требует вывести элементы xslt буквально, применяем -->
<!-- псевдоним xso -->
<xsl:namespace-alias stylesheet-prefix="xso" result-prefix="xsl"/>

<xsl:template match="xsl:stylesheet | xsl:transform">
  <xso:stylesheet>
     <!-- На первом проходе транслируем if-elsif-else -->
     <!-- и преобразуем xslx:loop в вызовы именованных шаблонов -->
     <xsl:apply-templates select="@* | node(  )"/>

     <!-- На втором проходе преобразуем xslx:loop в рекурсивные -->
     <!-- именованные шаблоны -->
     <xsl:apply-templates mode="loop-body" select="//xslx:loop"/>

  </xso:stylesheet>
</xsl:template>

<!-- Ищем элементы xslx:if, которым соответствуют элементы xslx:elsif
или xslx:else -->
<xsl:template match="xslx:if[following-sibling::xslx:else or
                             following-sibling::xslx:elsif]">
  <xsl:variable name="idIf" select="generate-id(  )"/>
  <xso:choose>
     <xso:when test="{@test}">
        <xsl:apply-templates select="@* | node(  )"/>
     </xso:when>
    <!-- Обрабатываем xsl:eslif и xslx:else в специальном режиме -->
    <!-- как часть xsl:choose. Нужно выбирать только такие элементы, -->
    <!-- для которых предшествующий элемент xslx:if совпадает с данным -->
   <!-- xslx:if -->
     <xsl:apply-templates
        select="following-sibling::xslx:else[
                  generate-id(preceding-sibling::xslx:if[1]) = $idIf] |
        following-sibling::xslx:elsif[
```

```
                    generate-id(preceding-sibling::xslx:if[1]) = $idIf]"
          mode="choose"/>
    </xso:choose>
</xsl:template>

<!-- Игнорируем xslx:elsif и xslx:else в нормальном режиме -->
<xsl:template match="xslx:elsif | xslx:else"/>

<!-- xslx:elsif превращается в xsl:when -->
<xsl:template match="xslx:elsif"  mode="choose">
 <xso:when test="{@test}">
    <xsl:apply-templates select="@* | node(  )"/>
 </xso:when>
</xsl:template>

<!-- xslx:else превращается в xsl:otherwise -->
<xsl:template match="xslx:else" mode="choose">
   <xso:otherwise>
      <xsl:apply-templates/>
   </xso:otherwise>
</xsl:template>

<!-- xslx:loop превращается в вызов именованного шаблона -->
<xsl:template match="xslx:loop">
   <!-- Каждому шаблону присваивается имя loop-N, где N - позиция -->
   <!-- данного цикла относительно предшествующих циклов на -->
   <!-- любом уровне -->
   <xsl:variable name="name">
      <xsl:text>loop-</xsl:text>
      <xsl:number count="xslx:loop" level="any"/>
   </xsl:variable>

   <xso:call-template name="{$name}">
      <xsl:for-each select="ancestor::xslx:loop">
         <xso:with-param name="{@param}" select="${@param}"/>
      </xsl:for-each>
      <xso:with-param name="{@param}" select="{@init}"/>
   </xso:call-template>

</xsl:template>

<!-- Режим loop-body используется на втором проходе. -->
<!-- Здесь генерируются рекурсивные шаблоны, реализующие цикл. -->
```

```
<xsl:template match="xslx:loop" mode="loop-body">
  <xsl:variable name="name">
    <xsl:text>loop-</xsl:text>
    <xsl:value-of select="position(  )"/>
  </xsl:variable>

  <xso:template name="{$name}">
      <!-- Если этот цикл является вложенным, то он должен -->
      <!--"видеть" параметры внешнего цикла, поэтому сгенерируем их -->
      <xsl:for-each select="ancestor::xslx:loop">
          <xso:param name="{@param}"/>
      </xsl:for-each>
      <!-- Параметр локального цикла -->
      <xso:param name="{@param}"/>
      <!-- Генерируем проверку завершения рекурсии -->
      <xso:if test="{@test}">
          <!-- Применить шаблон в нормальном режиме для обработки -->
          <!-- обращений к вложенным циклам, а все остальное просто -->
          <!-- скопировать. -->
          <xsl:apply-templates/>
          <!-- Это рекурсивный вызов, увеличивающий параметр цикла -->
          <xso:call-template name="{$name}">
             <xsl:for-each select="ancestor::xslx:loop">
                <xso:with-param name="{@param}" select="${@param}"/>
             </xsl:for-each>
             <xso:with-param name="{@param}" select="${@param} + {@incr}"/>
          </xso:call-template>
      </xso:if>
  </xso:template>
</xsl:template>

</xsl:stylesheet>
```

А вот результат применения этой таблицы стилей:

```
<xso:stylesheet xmlns:xso="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xslx="http://
www.ora.com/XSLTCookbook/
ExtendedXSLT" version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="foo">
    <xso:choose>
      <xso:when test="bar">
        <xsl:text>Нередко рядом с bar встречается и foo!</xsl:text>
```

```
    </xso:when>
    <xso:when test="baz">
      <xsl:text>Слово baz - верный признак сленга</xsl:text>
    </xso:when>
    <xso:otherwise>
      <xso:call-template name="loop-1">
        <xso:with-param name="i" select="0"/>
      </xso:call-template>
    </xso:otherwise>
  </xso:choose>
  <xso:call-template name="loop-2">
    <xso:with-param name="i" select="10"/>
  </xso:call-template>
  <xso:choose>
    <xso:when test="foo">
      <xsl:text>foo foo! Никто не говорит foo foo!</xsl:text>
    </xso:when>
    <xso:otherwise>
      <xsl:text>Ну и ладно!</xsl:text>
    </xso:otherwise>
  </xso:choose>
</xsl:template>
<xso:template name="loop-1">
  <xso:param name="i"/>
  <xso:if test="$i &lt; 10">
    <xsl:text>Гм, сказать-то нечего, но повторю это 10 раз.</xsl:text>
    <xso:call-template name="loop-1">
      <xso:with-param name="i" select="$i + 1"/>
    </xso:call-template>
  </xso:if>
</xso:template>
<xso:template name="loop-2">
  <xso:param name="i"/>
  <xso:if test="$i &gt;= 0">
    <xso:call-template name="loop-3">
      <xso:with-param name="i" select="$i"/>
      <xso:with-param name="j" select="10"/>
    </xso:call-template>
    <xso:call-template name="loop-2">
      <xso:with-param name="i" select="$i + -1"/>
    </xso:call-template>
  </xso:if>
</xso:template>
<xso:template name="loop-3">
```

```
      <xso:param name="i"/>
      <xso:param name="j"/>
      <xso:if test="$j &gt;= 0">
        <xsl:text>
</xsl:text>
        <xsl:value-of select="$i * $j"/>
        <xso:call-template name="loop-3">
          <xso:with-param name="i" select="$i"/>
          <xso:with-param name="j" select="$j + -1"/>
        </xso:call-template>
      </xso:if>
    </xso:template>
</xso:stylesheet>
```

Обсуждение

Ключом к генерации XSLT из XSLT является элемент `xsl:namespace-alias`. Не будь его, процессор не смог бы отличить XSLT-код исполняемой таблицы стилей от кода, который требуется выводить в результирующий файл. Генерация XSLT из XSLT бывает полезна и в других контекстах, например:

- *Включение кода в учебник.*
  Эта ситуация возникает, когда нужно включить фрагменты кода в документацию (обычно-то бывает наоборот), чтобы представить информацию в виде, удобном для человека, а не для компилятора.
- *Условные директивы include/import.*
  Если бы такой механизм был реализован в XSLT, то, скорее всего, потребовалось бы вводить неуклюжие расширения в модель программирования, аналогичные препроцессору программ на C.
- *Динамическое вычисление выражений XPath.*
  Тут речь идет о таблице стилей, которая генерирует XPath-код из внешнего источника и внедряет его в другую таблицу стилей, которая вычисляет выражения статически. Дополнительный уровень косвенности позволяет эмулировать динамическое поведение. Часто XPath-выражения включаются в некий документ. Например, можно встретить такое описание таблицы:

```
<table of="person">
    <column label="Firstname" content="name/firstname" />
    <column label="Surname" content="name/surname" />
    <column label="Age" content="@age" type="number" />
    <sort select="Surname, Firstname, Age" />
</table>
```

Таблицу, описываемую таким XML-документом, проще сгенерировать путем создания XSLT-кода по спецификации таблицы и последующего применения его

к данным, чем путем интерпретации этой спецификации в той же таблице стилей, которая используется для обработки данных.

См. также

Оливер Беккер написал похожий компилятор циклов в XSLT, который проверяет также и семантику (*http://www.informatik.hu-berlin.de/~obecker/XSLT/#loop-compiler*).

Глава 13. Рецепты применения XSLT в вертикальных приложениях

Новоприбывший недоумевает,
есть ли какой-нибудь секретный знак
или слово, которое позволит проникнуть
в его тайны...например,
знать, что положить в общий котел.

Из рецензии на книгу
Potluck: Stories That Taste Like Hawaii

13.0. Введение

Эта глава отличается от прочих тем, что приведенные в ней примеры – это мини-приложения XSLT в самых разных областях («общий котел», если хотите). Многие примеры связаны с теми или иными коммерческими программами. Вводя поддержку XML, производители программного обеспечения открывают такие возможности применения своих продуктов, о которых и сами не подозревали (или не собирались реализовывать).

Корпорация Microsoft – один из производителей, присоединившихся к партии сторонников XML. В последние версии Microsoft Visio (версия 10.0) Excel (Office XP версия 10.0) добавлена поддержка вывода в формате XML. Visio – это коммерческий пакет векторной графики, и применяемый в нем формат XML (он называется Visio VDX) специфичен именно для этого продукта. Джон Брин (John Breen) проделал прекрасную работу по конвертированию этого формата в SVG (Scalable Vector Graphics). Его код составляет содержание рецепта 13.1.

Microsoft Excel позволяет сохранять электронные таблицы в формате XML. К сожалению, схема буквально повторяет структуру электронной таблицы Excel. В рецепте 13.2 показано, как преобразовать ее к более удобному виду.

Тематические карты (Topic Maps) – новая, активно развиваемая XML-технология моделирования знаний с целью сделать опубликованную в Web информацию более доступной людям и компьютерам. XTM – открытый стандарт представления тематических карт в формате XML. Разработчики программного обеспечения моделируют знания о системе с помощью *унифицированного языка моделирования* (Unified Modeling Language – UML). Для UML определен собственный стандарт представления в формате XML, который называется XML Metadata Interchange (XMI). Хотя UML и тематические карты рассчитаны на разные аудитории, язык UML достаточно выразителен для описания концепций,

обслуживаемых стандартом Topic Maps, при соблюдении определенных соглашений. Поскольку UML существует дольше, чем Topic Maps, разработанные для него инструменты достигли большей зрелости. В рецепте 13.3 показано, как можно преобразовать XMI-документ, созданный популярной программой для работы с UML (Rational Rose), в формат XTM Topic Maps[1].

Одна из самых полезных особенностей языка XTM – возможность генерировать сайты. В рецепте 13.4 рассматривается такое приложение тематических карт. Этот рецепт предложил Никита Огивецкий (Nikita Ogievetsky), он основан на результатах его работы над каркасом, используемым в сайте Cogitative Topic Maps Web Site (CTW).

И завершим мы эту главу рассмотрением протокола SOAP (Simple Object Access Protocol – простой протокол доступа к объектам), который применяется для создания Web-сервисов и специфицирован консорциумом W3C. Этот протокол позволяет программным системам обмениваться между собой стандартизованными XML-сообщениями. Мы рассмотрим язык WSDL (Web Service Definition Language – язык определения Web-сервисов), тесно связанный с SOAP, и покажем, как преобразовать WSDL-описание в документацию, понятную человеку.

Примеры в этой главе довольно объемные, полный исходный текст можно найти на странице *http://www.oreilly.com/catalog/xsltckbk/*.

13.1. Преобразование документов из формата Visio VDX в формат SVG

Задача

Требуется преобразовать файл в формате Microsoft Visio XML (VDX) в более переносимый формат SVG.

Решение

Показанное ниже решение предложил Джон Брин. Основные элементы Visio отображаются на элементы SVG, как показано в таблице 13.1.

***Таблица 13.1.** Отображение элементов Visio на элементы SVG*

Элемент Visio	Элемент SVG
`VisioDocument/Colors/ColorEntry`	Цвет
`VisioDocument/Stylesheets/StyleSheet`	Стиль CSS
`VisioDocument/Pages/Page`	`Svg`
`Page/Shapes/Shape`	`G`
`Shapes/Shape/@NameU`	`@id`
`Shapes/Shape/Xform`	`transform`
`XForm/PinX and XForm/PinX`	`translate( )`
`Shapes/Shape/Fill/FillForegnd`	`@fill` (с `lookup`)

[1] Можно было бы назвать этот рецепт Acronym Conversion Software (ACS – программа конвертации акронимов).

Таблица 13.1. *Отображение элементов Visio на элементы SVG (окончание)*

Элемент Visio	Элемент SVG
Shapes/Shape/Geom	path
Shape/Geom/MoveTo	@d "M"
Shape/Geom/LineTo	@d "L"
Shape/Geom/NoFill(0)	@d "z"

В этом разделе мы рассмотрим только главную таблицу стилей и отдельные части включаемых таблиц. Полностью исходный код вместе с примерами можно найти по адресу *http://sourceforge.net/projects/vdxtosvg/*.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:v="urn:schemas-microsoft-com:office:visio"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:math="java.lang.Math"
  xmlns:jDouble="java.lang.Double"
  xmlns:saxon="http://icl.com/saxon"
  exclude-result-prefixes="v math saxon jDouble"
  xmlns="http://www.w3.org/2000/svg"
  version="1.0">

  <xsl:output method="xml"
    version="1.0"
    omit-xml-declaration="no"
    media-type="image/svg+xml"
    encoding="iso-8859-1"
    indent="yes"
    cdata-section-elements="style"
    doctype-public="-//W3C//DTD SVG 1.0//EN"
    doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
  />
```

Параметр `pageNumber` задает номер страницы, извлекаемой из VDX-файла, а параметр `userScale` – коэффициент масштабирования для перевода из единиц Visio в единицы пользователя:

```
  <xsl:param name="pageNumber" select="1"/>
  <xsl:param name="userScale"  select="100"/>

  <!-- =  =  Переменные (т.е. константы) =  =  =  =  =  =  =  =  = -->
  <!-- Таблица отображения цветов -->
  <xsl:variable name="Colors"
      select="//v:Colors[position(  )=1]/v:ColorEntry"/>

  <!-- Обрабатываемая страница -->
  <xsl:variable name="Page"
```

```
    select="/v:VisioDocument/v:Pages/v:Page[number($pageNumber)]"/>

<!-- Главные шаблоны -->
<xsl:variable name="Masters"
    select="//v:Masters[position(  )=1]/v:Master"/>

<!-- viewBox Master -->
<xsl:variable name="viewBoxMaster"
    select="$Masters[@NameU='viewBox']"/>

<!-- Отношение высоты шрифта к ширине (поправочный коэффициент) -->
<xsl:variable name="fontRatio"
    select="2"/>

<!-- Число Pi (в SVG используются градусы, в Visio радианы) -->
<xsl:variable name="pi" select="3.14159265358979323846264338327"/>
```

Таблица стилей разбита на несколько компонентов, которые включаются командой `include`. Часть из них будет рассмотрена ниже. Используются расширения на языке JavaScript. Впрочем, если имеющийся у вас процессор XSLT не поддерживает JavaScript, код все равно будет работать, хотя форматирование будет не таким красивым.

```
<!-- Включаемые файлы -->
<xsl:include href="visio-style.xsl"/>
<xsl:include href="visio-text.xsl"/>
<xsl:include href="visio-masters.xsl"/>
<xsl:include href="visio-nurbs.xsl"/>

<!-- Сценарии -->
<xsl:template name="required-scripts">
       <script xlink:href="wordwrap.js" type="text/ecmascript"/>
</xsl:template>

<xsl:template match="/v:VisioDocument">
   <xsl:apply-templates
      select="$Page"/>
</xsl:template>
```

Страница Visio отображается на графику в формате SVG. Извлеченная из документа Visio информация определяет, как оптимально расположить графику в области просмотра:

```
<!-- =  =  =  =  = Страница =  =  =  =  =  =  =  =  =  =  =  =-->
<xsl:template match="v:Page">
   <xsl:message>
    <xsl:value-of select="@NameU"/>
```

```
      </xsl:message>
      <svg id="{@NameU}">
      <xsl:attribute name="xml:space">
         <xsl:value-of select="'preserve'"/>
      </xsl:attribute>
      <xsl:choose>
         <!-- Использовать viewBox с именем 'default', если таковой
имеется -->
      <xsl:when test="//v:Shape[@Master=$viewBoxMaster/@ID
                     and @NameU='default'][1]">
         <xsl:for-each
            select="//v:Shape[@Master=$viewBoxMaster/@ID
                     and @NameU='default']">
            <xsl:attribute name="viewBox">
             <xsl:value-of select="concat(
                                 v:XForm/v:PinX*$userScale, ' ',
                                 -v:XForm/v:PinY*$userScale, ' ',
                                 v:XForm/v:Width*$userScale, ' ',
                                 v:XForm/v:Height*$userScale)"/>
               </xsl:attribute>
            </xsl:for-each>
         </xsl:when>
         <!-- Иначе расположить в центре листа -->
         <xsl:otherwise>
            <xsl:attribute name="viewBox">
             <xsl:value-of select="concat('0 ',
                           -v:PageSheet/v:PageProps/v:PageHeight
                                 *$userScale, ' ',
                           v:PageSheet/v:PageProps/v:PageWidth
                                 *$userScale, ' ',
                           v:PageSheet/v:PageProps/v:PageHeight
                                 *$userScale)"/>
               </xsl:attribute>
         </xsl:otherwise>
      </xsl:choose>
      <xsl:call-template name="required-scripts"/>
      <xsl:call-template name="predefined-pattern-fgnds"/>
      <xsl:call-template name="predefined-markers"/>
```

Отсюда начинается собственно преобразование. Для начала обработаем элементы Visio `StyleSheet` и преобразуем их в правила Cascading Style Sheet. Затем преобразуем все формы в эквивалентное представление на SVG:

```
      <xsl:apply-templates select="../../v:StyleSheets"/>
      <xsl:apply-templates select="v:Shapes/v:Shape"/>
```

```
    </svg>
</xsl:template>

<!-- = = = = = StyleSheets = = = = = = = = = = = = = -->
<xsl:template match="v:StyleSheets">
    <defs>
        <xsl:for-each select="v:StyleSheet">
            <!-- Стиль линии -->
            <style id="ss-line-{@ID}" type="text/css">
            <xsl:text>*.ss-line-</xsl:text><xsl:value-of select="@ID"/>
            <xsl:text> { </xsl:text>
            <xsl:call-template name="recursive-line-style">
                <xsl:with-param name="ss" select="."/>
            </xsl:call-template>
            <xsl:text> }</xsl:text>
            </style>
            <!-- Стиль заливки -->
            <style id="ss-fill-{@ID}" type="text/css">
            <xsl:text>*.ss-fill-</xsl:text><xsl:value-of select="@ID"/>
            <xsl:text> { </xsl:text>
            <xsl:call-template name="recursive-fill-style">
                <xsl:with-param name="ss" select="."/>
            </xsl:call-template>
            <xsl:text> }</xsl:text>
            </style>
            <!-- Стиль текста -->
            <style id="ss-text-{@ID}" type="text/css">
            <xsl:text>*.ss-text-</xsl:text><xsl:value-of select="@ID"/>
            <xsl:text> { </xsl:text>
            <xsl:call-template name="recursive-text-style">
                <xsl:with-param name="ss" select="."/>
            </xsl:call-template>
            <xsl:text> } </xsl:text>
            </style>
        </xsl:for-each>
    </defs>
</xsl:template>

<!-- Рекурсивно обрабатываем наследование StyleSheet -->
<xsl:template name="recursive-line-style">
    <xsl:param name="ss"/>
    <xsl:if test="$ss/@LineStyle">
        <xsl:call-template name="recursive-line-style">
            <xsl:with-param name="ss"
```

```
                select="$ss/../v:StyleSheet[@ID=$ss/@LineStyle]"/>
            </xsl:call-template>
        </xsl:if>
        <xsl:apply-templates select="$ss/v:Line" mode="style"/>
    </xsl:template>

    <xsl:template name="recursive-fill-style">
        <xsl:param name="ss"/>
        <xsl:if test="$ss/@FillStyle">
            <xsl:call-template name="recursive-fill-style">
                <xsl:with-param name="ss"
                    select="$ss/../v:StyleSheet[@ID=$ss/@FillStyle]"/>
            </xsl:call-template>
        </xsl:if>
        <xsl:apply-templates select="$ss/v:Fill" mode="style"/>
    </xsl:template>

    <xsl:template name="recursive-text-style">
        <xsl:param name="ss"/>
        <xsl:if test="$ss/@TextStyle">
            <xsl:call-template name="recursive-text-style">
                <xsl:with-param name="ss"
                    select="$ss/../v:StyleSheet[@ID=$ss/@TextStyle]"/>
            </xsl:call-template>
        </xsl:if>
        <xsl:apply-templates select="$ss/v:Char|$ss/v:Para" mode="style"/>
    </xsl:template>

    <!-- Этот шаблон возвращает строку, описывающую стиль линии -->
    <xsl:template match="v:Line" mode="style">
        <xsl:for-each select="v:LineWeight">
            <xsl:text>stroke-width:</xsl:text>
            <xsl:value-of select=". * $userScale"/><xsl:text>;</xsl:text>
        </xsl:for-each>
        <xsl:for-each select="v:LineColor">
            <xsl:choose>
                <xsl:when test="../v:LinePattern > 0">
                    <xsl:text>stroke:</xsl:text>
                    <xsl:call-template name="lookup-color">
                        <xsl:with-param name="c_el" select="."/>
                    </xsl:call-template>
                </xsl:when>
                <xsl:when test="../v:LinePattern = 0">
                    <xsl:text>stroke:none</xsl:text>
```

```
            </xsl:when>
         </xsl:choose>
         <xsl:text>;</xsl:text>
      </xsl:for-each>
      <xsl:for-each select="v:EndArrow">
         <xsl:choose>
            <xsl:when test=". = 0">
               <xsl:value-of select="string('marker-end:none;')"/>
            </xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="concat('marker-end:url(#EndArrow-', .,
                                            '-', ../v:EndArrowSize, ');')"/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:for-each>
      <xsl:apply-templates select="v:LinePattern[. &gt; 1]" mode="style"/>
</xsl:template>

<!-- Этот шаблон возвращает строку, описывающую стиль заливки -->
<xsl:template match="v:Fill" mode="style">
      <xsl:for-each select="v:FillForegnd">
         <xsl:choose>
            <xsl:when test="../v:FillPattern = 1">
               <xsl:text>fill:</xsl:text>
               <xsl:call-template name="lookup-color">
                  <xsl:with-param name="c_el" select="."/>
               </xsl:call-template>
            </xsl:when>
            <xsl:when test="../v:FillPattern = 0">
               <xsl:text>fill:none</xsl:text>
            </xsl:when>
            <xsl:otherwise>
               <xsl:text>fill:url(#</xsl:text>
               <xsl:value-of select="generate-id(../..)"/>
               <xsl:text>-pat)</xsl:text>
            </xsl:otherwise>
         </xsl:choose>
         <xsl:text>;</xsl:text>
      </xsl:for-each>
</xsl:template>

<!-- Этот шаблон возвращает строку, описывающую стиль текста -->
<xsl:template match="v:Char|v:Para" mode="style">
      <xsl:for-each select="v:Color">
```

```
            <!-- Не думаю, что Visio обрабатывает залитые символы -->
            <xsl:text>stroke:none</xsl:text>
            <xsl:text>;fill:</xsl:text>
            <xsl:call-template name="lookup-color">
               <xsl:with-param name="c_el" select="."/>
            </xsl:call-template>
            <xsl:text>;</xsl:text>
         </xsl:for-each>
         <xsl:for-each select="v:Size">
            <xsl:text>font-size:</xsl:text>
            <xsl:value-of select=". * $userScale"/><xsl:text>;</xsl:text>
         </xsl:for-each>
         <xsl:for-each select="v:HorzAlign">
            <xsl:text>text-anchor:</xsl:text>
            <xsl:choose>
               <xsl:when test="(. = 0) or (. = 3)">
                  <xsl:text>start</xsl:text>
               </xsl:when>
               <xsl:when test=". = 1">
                  <xsl:text>middle</xsl:text>
               </xsl:when>
               <xsl:when test=". = 2">
                  <xsl:text>end</xsl:text>
               </xsl:when>
            </xsl:choose>
            <xsl:text>;</xsl:text>
         </xsl:for-each>
      </xsl:template>

      <!-- Все остальные элементы StyleSheet игнорируем -->
      <xsl:template match="*[parent::v:StyleSheet]" priority="-100"/>
```

В следующем фрагменте формы отображаются на эквивалентные конструкции SVG. Обратите внимание, как можно ассоциировать формы с главными шаблонами в документе Visio. Можно считать, что главный шаблон – это образец, от которого нарисованная на странице форма наследует атрибуты и поведение.

По умолчанию каждая форма Visio транслируется в элемент `<g>`, так как она может содержать графику и текст. Напомним, что в SVG элемент `<g>` описывает группу графических элементов, имеющих общие стилистические характеристики.

`SvgElement` – это свойство Visio, которое пользователь может присоединить к форме для задания специальной обработки данным транслятором. Например, в качестве `svgElement` можно указать любой другой контейнерный элемент SVG, скажем `defs`. Таким способом можно запретить рисование некоторых форм, например, путей для элементов `animateMotion`. То есть сослаться на путь в SVG-файле можно, а отображаться он не будет.

Одно из основных назначений элемента animateMotion – обозначить специальные формы, которые транслируются в элементы, не имеющие аналогов в Visio, например, animate, animateMotion и viewBox. Эти элементы обрабатываются таблицей стилей *visio-master.xsl*, которую мы рассмотрим ниже.

```
 <!-- = = = = Форма = = = = = = = = = = = = = = = = -->
<xsl:template match="v:Shape">

 <xsl:variable name="master"
          select="/v:VisioDocument//v:Masters[1]/
                 v:Master[@ID=current(  )/@Master]"/>

 <xsl:variable name="svgElement">
   <xsl:choose>
     <!-- Проверить наличие специального свойства svgElement у формы ... -->
     <xsl:when test="./v:Prop/v:Label[.='svgElement']">
        <xsl:value-of
           select="./v:Prop/v:Label[.='svgElement']/../v:Value"/>
     </xsl:when>
     <!-- ... и у главного шаблона -->
     <xsl:when test="@Master and
        $master//v:Prop/v:Label[.='svgElement']">
        <xsl:value-of
          select="$master//v:Prop/v:Label[.='svgElement']/../v:Value"/>
     </xsl:when>

     <!-- Простой случай: форма отображается на группу svg - элемент g -->
     <xsl:otherwise>
        <xsl:value-of select="'g'"/>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:variable>

<xsl:choose>

   <xsl:when test="@Master and string($svgElement)
             and contains($specialMasters, $svgElement)">
     <xsl:call-template name="choose-special-master">
        <xsl:with-param name="master" select="$master"/>
        <xsl:with-param name="masterElement" select="$svgElement"/>
     </xsl:call-template>
   </xsl:when>

   <xsl:when test="($svgElement = 'defs') or ($svgElement = 'g') or
          ($svgElement = 'symbol')">
```

```
        <xsl:choose>
           <xsl:when test="v:Hyperlink">
              <!-- Погрузить форму в элемент 'a' -->
              <!-- Это минимальная реализация. Не поддерживаются -->
              <!-- множественные ссылки, подадрес и т.д. -->
               <a xlink:title="{v:Hyperlink/v:Description}"
                   xlink:href="{v:Hyperlink/v:Address}">
                   <xsl:if test="v:Hyperlink/v:NewWindow">
                      <xsl:attribute name="show">
                        <xsl:value-of select="new"/>
                      </xsl:attribute>
                   </xsl:if>
                   <xsl:element name="{$svgElement}">
                      <xsl:call-template name="userShape"/>
                   </xsl:element>
               </a>
           </xsl:when>
           <xsl:otherwise>
               <xsl:element name="{$svgElement}">
                   <xsl:call-template name="userShape"/>
               </xsl:element>
           </xsl:otherwise>
        </xsl:choose>
      </xsl:when>
    </xsl:choose>
  </xsl:template>
```

Далее обычные формы, созданные пользователем, отображаются на элементы SVG, как описано в таблице 13.1.

```
  <!-- Обработка обычных 'пользовательских' форм -->
  <xsl:template name="userShape">
    <xsl:variable name="master"
      select="/v:VisioDocument/v:Masters
        /v:Master[(@ID=current(  )/@Master)
                  and (current(  )/@Type != 'Group')]
        /v:Shapes/v:Shape |
        /v:VisioDocument/v:Masters
        /v:Master[@ID=current(  )
        /ancestor::v:Shape[@Master]/@Master]
        //v:Shape[@ID=current(  )/@MasterShape] | ."/>

    <xsl:call-template name="setIdAttribute"/>

    <xsl:attribute name="class">
```

```
      <xsl:for-each select="($master[@LineStyle])[last(  )]">
        <xsl:text> ss-line-</xsl:text>
        <xsl:value-of select="@LineStyle"/>
      </xsl:for-each>
      <xsl:for-each select="($master[@FillStyle])[last(  )]">
        <xsl:text> ss-fill-</xsl:text>
        <xsl:value-of select="@FillStyle"/>
      </xsl:for-each>
    </xsl:attribute>
    <xsl:attribute name="style">
      <xsl:for-each select="$master">
        <xsl:apply-templates select="./v:Line" mode="style"/>
        <xsl:apply-templates select="./v:Fill" mode="style"/>
      </xsl:for-each>
    </xsl:attribute>
    <xsl:for-each select="v:XForm">
      <xsl:call-template name="transformAttribute">
      </xsl:call-template>
    </xsl:for-each>
    <!-- Это нужно для создания специального образца -->
    <xsl:apply-templates select="v:Fill" mode="Shape"/>
    <xsl:for-each select="v:Geom">
      <xsl:apply-templates select="v:Ellipse"/>
      <xsl:if test="v:MoveTo or v:LineTo">
        <xsl:call-template name="pathElement"/>
      </xsl:if>
    </xsl:for-each>
    <xsl:for-each select="($master/v:Text)[last(  )]">
      <xsl:apply-templates select="."/>
    </xsl:for-each>

    <xsl:apply-templates select="v:Shapes/v:Shape"/>

    <!-- Добавить элементы, соответствующие свойствам -->
    <xsl:for-each select="v:Prop">
      <xsl:choose>
        <xsl:when test="starts-with(v:Label, 'svg-element')">
          <!-- Криво - в будущем, возможно, будет убрано -->
          <xsl:value-of disable-output-escaping="yes" select="v:Value"/>
        </xsl:when>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>

  <xsl:template match="v:Ellipse">
```

```
    <!-- Здесь у трансляции есть ограничения. Предполагается, что оси
    параллельны осям x и y, а левый нижний угол охватывающего
    прямоугольника находится в начале координат (похоже, именно так
    Visio рисует по умолчанию). -->
    <ellipse id="ellipse-{generate-id(ancestor::v:Shape[1])}"
      cx="{v:X*$userScale}" cy="{-v:Y*$userScale}"
      rx="{v:X*$userScale}" ry="{v:Y*$userScale}"/>
  </xsl:template>

  <!-- = = = = Вспомогательные шаблоны = = = = = = = = = = -->

  <!-- Поиск значения цвета в элементе Colors -->
  <xsl:template name="lookup-color">
    <xsl:param name="c_el"/>
    <xsl:choose>
      <xsl:when test="starts-with($c_el, '#')">
        <xsl:value-of select="$c_el"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$Colors[@IX=string($c_el)]/@RGB"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Если у элемента Visio есть имя, берем его в качестве идентификатора формы; в противном случае вызываем `generate-id()`:

```
  <xsl:template name="setIdAttribute">
    <xsl:attribute name="id">
       <xsl:choose>
       <xsl:when test="@NameU">
         <xsl:value-of select="@NameU"/>
       </xsl:when>
       <xsl:otherwise>
          <xsl:value-of select="generate-id(.)"/>
       </xsl:otherwise>
       </xsl:choose>
    </xsl:attribute>
  </xsl:template>

  <!-- Транслировать элемент XForm в атрибут transform -->
  <xsl:template name="transformAttribute">
    <xsl:attribute name="transform">
       <xsl:text>translate(</xsl:text>
       <xsl:value-of select="concat((v:PinX - v:LocPinX)*$userScale,
                             ',', -(v:PinY - v:LocPinY)*$userScale)"/>
```

```
      <xsl:if test="v:Angle != 0">
        <xsl:text>) rotate(</xsl:text>
        <xsl:value-of select="-v:Angle*180 div $pi"/>
        <xsl:value-of select="concat(',', v:LocPinX*$userScale,
                              ',', -v:LocPinY*$userScale)"/>
      </xsl:if>
      <xsl:text>)</xsl:text>
    </xsl:attribute>
</xsl:template>
```

Элементы Vision `Geom` транслируются в пути. По большей части, отображение прямолинейно. Исключение составляет обработка неоднородных рациональных B-сплайнов (Non-Uniform Rational B-Splines – NURBS), которая делегируется набору специальных шаблонов в файле *visio-nurbs.xsl* (имеется в полном дистрибутиве).

```
<!-- Транслировать элемент Geom element в элемент path -->
<xsl:template name="pathElement">
  <xsl:variable name="pathID">
    <xsl:text>path-</xsl:text>
    <xsl:choose>
      <xsl:when test="ancestor::v:Shape[1]/@NameU">
        <xsl:value-of select="ancestor::v:Shape[1]/@NameU"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="generate-id(ancestor::v:Shape[1])"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
  <path id="{$pathID}">
    <xsl:attribute name="d">
      <xsl:for-each select="v:*">
        <xsl:choose>
          <xsl:when test="name(  ) = 'MoveTo'">
            <xsl:value-of select="concat('M', v:X*$userScale,
                                  ',', -v:Y*$userScale, ' ')"/>
      </xsl:when>
      <xsl:when test="name(  ) = 'LineTo'">
        <xsl:value-of select="concat('L', v:X*$userScale,
                              ',', -v:Y*$userScale, ' ')"/>
          </xsl:when>
          <xsl:when test="name(  ) = 'EllipticalArcTo'">
        <!-- Если тригонометрические функции недоступны, то дуга
        будет представлена двумя отрезками прямых -->
            <xsl:choose>
```

```
                <xsl:when test="function-available('math:atan2')">
                    <xsl:call-template name="ellipticalArcPath"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="concat('L', v:A*$userScale,
                                          ',', -v:B*$userScale,
                                          ' L', v:X*$userScale,
                                          ',', -v:Y*$userScale, ' ')"/>
                </xsl:otherwise>
              </xsl:choose>
            </xsl:when>
            <xsl:when test="(name(  ) = 'NoFill') or (name(  ) = 'NoLine') or
                            (name(  ) = 'NoShow') or (name(  ) = 'NoSnap')">
          <!-- Игнорируются -->
        </xsl:when>
        <xsl:when test="name(  ) = 'NURBSTo'">
          <xsl:call-template name="NURBSPath"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:message>
            <xsl:text>Предупреждение: обнаружена неподдерживаемая
            команда path:</xsl:text>
            <xsl:value-of select="name(  )"/>
            <xsl:text>; replacing with LineTo</xsl:text>
          </xsl:message>
          <xsl:value-of select="concat('L', v:X*$userScale,
                                ',', -v:Y*$userScale, ' ')"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:attribute>
  <xsl:if test="v:NoFill = 1">
     <xsl:attribute name="fill"><xsl:text>none</xsl:text></xsl:attribute>
    </xsl:if>
  </path>
</xsl:template>

  <!-- Этот шаблон вычисляет строку path для дуги эллипса -->

  <xsl:template name="ellipticalArcPath">

  <!-- Нарисовать дугу, зная углы между текущей точкой  и точками (X,Y)
  и (A,B) -->

   <!-- TODO: найти более удачный способ убедиться в том,
```

```
<!-- что предшествующий брат — элемент, относящийся к рисованию -->

   <xsl:variable name="lastX"
      select="preceding-sibling::*[1]/v:X"/>
   <xsl:variable name="lastY"
      select="preceding-sibling::*[1]/v:Y"/>
   <xsl:variable name="angle"
      select="math:atan2(v:Y - $lastY, v:X - $lastX)
              - math:atan2(v:B - $lastY, v:A - $lastX)"/>
   <xsl:variable name="sweep">
      <xsl:choose>
     <xsl:when test="$angle &gt; 0
                      and math:abs($angle) &lt; 180">
       <xsl:value-of select='0'/>
     </xsl:when>
     <xsl:when test="$angle &lt; 0
                      and math:abs($angle) &gt; 180">
       <xsl:value-of select='0'/>
     </xsl:when>
     <xsl:otherwise>
       <xsl:value-of select='1'/>
     </xsl:otherwise>
      </xsl:choose>
   </xsl:variable>
   <xsl:value-of select="concat('A',
                  (preceding-sibling::*[1]/v:X - v:X)*$userScale, ',',
                  (preceding-sibling::*[1]/v:Y - v:Y)*$userScale, ' ',
                  v:C,  ' 0,', $sweep, ' ', v:X*$userScale, ',',
                  -v:Y*$userScale, ' ')"/>
 </xsl:template>

</xsl:stylesheet>
```

Обсуждение

В последней версии Visio поддерживается вывод в формате SVG. Тем не менее этот рецепт остается полезным, если у вас установлена старая версия Visio или преобразование, выполняемое в текущей версии, вас не устраивает. Понятно, что SVG – недостаточно развитый формат, чтобы точно представить все конструкции Visio, но аппроксимация довольно близкая. Простые диаграммы Visio можно транслировать в SVG почти без искажений. В более сложных случаях, возможно, придется подправить результат в SVG-редакторе. В замечаниях к версии, включенных в состав полного дистрибутива Visio, все отсутствующие возможности перечислены. На рис. 13.1 показан пример SVG-файла, сгенерированного из диаграммы Visio; видно, что он отображается практически без изъянов.

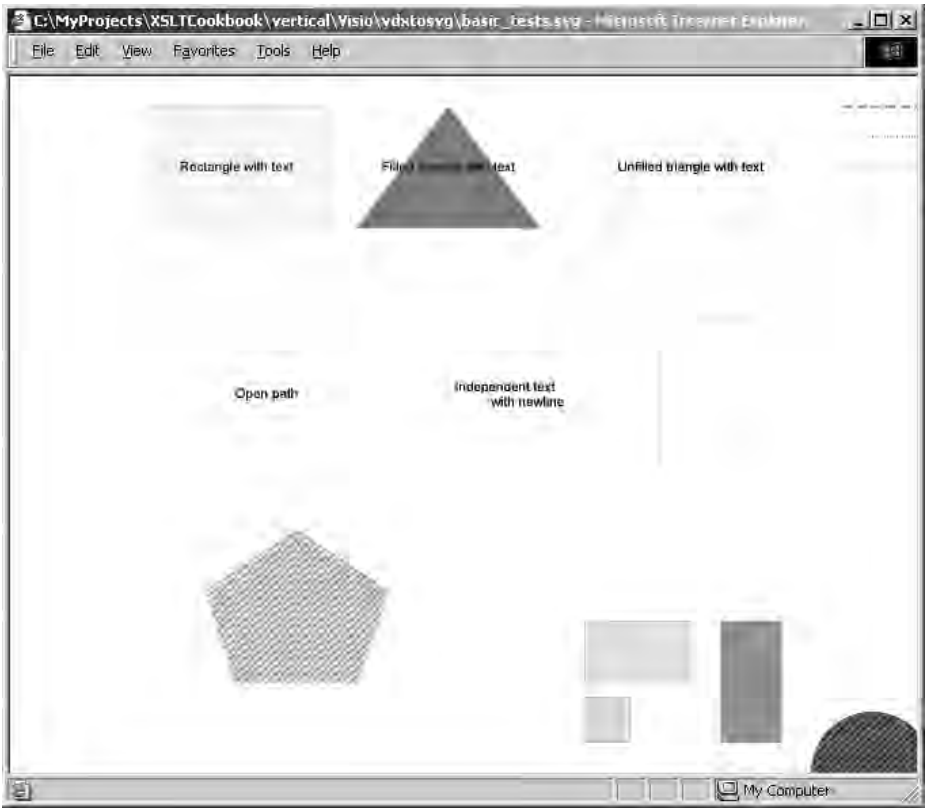


Рис. 13.1. Простые SVG-формы и текст, полученные преобразованием файла Visio

Урок, который можно извлечь из этого примера, отнюдь не ограничивается деталями форматов Visio VDX , SVG и специальными приемами, которые применил автор. Интересно, что это сложное преобразование было первым опытом автора по работе с XSLT. И этот факт многое говорит о выразительности парадигмы преобразований, принятой в XSLT.

См. также

Исходный код и ряд демонстраций можно найти на сайте Source Forge по адресу *http://sourceforge.net/projects/vdxtosvg/*.

13.2. Работа с электронными таблицами в формате Excel XML

Задача

Требуется экспортировать данные из Excel в XML, но не в том формате, который предлагает Microsoft.

Решение

XSLT 1.0

Пусть есть такая электронная таблица Excel:

Date	Price	Volume
20010817	61.88	260163
20010820	62.7	241859
20010821	60.78	233989
20010822	60.66	387444

При выводе в формате XML (в версиях Excel XP или 2003) получается следующий результат:

```
<?xml version="1.0"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
xmlns:o="urn:schemas-microsoft-com:office:office" xmlns:x="urn:schemas-
microsoft-com:office:excel"
xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet" xmlns:html="http://
www.w3.org/TR/REC-html40">
  <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
    <Author>Salvatore R. Mangano</Author>
    <LastAuthor>Salvatore R. Mangano</LastAuthor>
    <Created>2002-08-18T00:43:49Z</Created>
    <LastSaved>2002-08-18T02:19:21Z</LastSaved>
    <Company>Descriptix</Company>
    <Version>10.3501</Version>
  </DocumentProperties>
  <OfficeDocumentSettings xmlns="urn:schemas-microsoft-com:office:office">
    <DownloadComponents/>
    <LocationOfComponents HRef="/"/>
  </OfficeDocumentSettings>
  <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
    <WindowHeight>9915</WindowHeight>
    <WindowWidth>10140</WindowWidth>
    <WindowTopX>240</WindowTopX>
    <WindowTopY>255</WindowTopY>
    <ProtectStructure>False</ProtectStructure>
    <ProtectWindows>False</ProtectWindows>
  </ExcelWorkbook>
  <Styles>
    <Style ss:ID="Default" ss:Name="Normal">
      <Alignment ss:Vertical="Bottom"/>
      <Borders/>
      <Font/>
```

```
    <Interior/>
    <NumberFormat/>
    <Protection/>
  </Style>
</Styles>
<Worksheet ss:Name="msft">
  <Table ss:ExpandedColumnCount="3" ss:ExpandedRowCount="5" x:FullColumns="1"
    x:FullRows="1">
     <Row>
    <Cell>
       <Data ss:Type="String">Date</Data>
    </Cell>
    <Cell>
       <Data ss:Type="String">Price</Data>
    </Cell>
    <Cell>
       <Data ss:Type="String">Volume</Data>
      </Cell>
     </Row>
     <Row>
    <Cell>
       <Data ss:Type="Number">20010817</Data>
    </Cell>
    <Cell>
       <Data ss:Type="Number">61.88</Data>
    </Cell>
    <Cell>
       <Data ss:Type="Number">260163</Data>
    </Cell>
     </Row>
     <Row>
    <Cell>
       <Data ss:Type="Number">20010820</Data>
    </Cell>
    <Cell>
       <Data ss:Type="Number">62.7</Data>
    </Cell>
    <Cell>
       <Data ss:Type="Number">241859</Data>
    </Cell>
     </Row>
     <Row>
    <Cell>
       <Data ss:Type="Number">20010821</Data>
```

```
      </Cell>
      <Cell>
         <Data ss:Type="Number">60.78</Data>
      </Cell>
      <Cell>
         <Data ss:Type="Number">233989</Data>
      </Cell>
       </Row>
       <Row>
      <Cell>
         <Data ss:Type="Number">20010822</Data>
      </Cell>
      <Cell>
         <Data ss:Type="Number">60.66</Data>
      </Cell>
      <Cell>
         <Data ss:Type="Number">387444</Data>
      </Cell>
       </Row>
    </Table>
    <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
      <Selected/>
      <Panes>
         <Pane>
            <Number>3</Number>
            <ActiveRow>11</ActiveRow>
            <ActiveCol>5</ActiveCol>
         </Pane>
      </Panes>
      <ProtectObjects>False</ProtectObjects>
      <ProtectScenarios>False</ProtectScenarios>
    </WorksheetOptions>
  </Worksheet>
</Workbook>
```

Но это совершенно не то, что вам нужно!

В настоящем примере мы преобразуем файл из формата Excel XML в более простой. Во многих создаваемых в Excel таблицах первая строка содержит названия колонок, а последующие строки – данные в этих колонках.

Очевидное решение – преобразовать названия колонок в имена элементов, а остальные ячейки – в содержимое этих элементов. Не хватает только имен элемента верхнего уровня и элементов, содержащих данные каждой строки. В следующей таблице стилей эти имена будут передаваться в качестве параметров с очевидными умолчаниями. Некоторые полезные метаданные будут преобразованы в комментарии, а специфичная для Excel информация отброшена. Мы введем еще

несколько параметров для повышения общности преобразования, в частности, информацию о том, какая строка содержит названия колонок, с какой строки начинаются данные и что делать с пустыми ячейками.

```
<?xml version="1.0" encoding="UTF-8"?>
              <xsl:stylesheet version="1.0"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
              xmlns:o="urn:schemas-microsoft-com:office:office"
              xmlns:x="urn:schemas-microsoft-com:office:excel"
              xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet">

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- Имя элемента верхнего уровня -->
  <xsl:param name="topLevelName" select=" 'Table' "/>
  <!-- Имя строки -->
  <xsl:param name="rowName" select=" 'Row' "/>
  <!-- Название используемого пространства имен -->
  <xsl:param name="namespace"/>
  <!-- Префикс используемого пространства имен -->
  <xsl:param name="namespacePrefix"/>
  <!-- Какой символ использовать вместо пробела в названиях колонок -->
  <xsl:param name="wsSub" select="'_'"/>
  <!-- Какая строка содержит названия колонок -->
  <xsl:param name="colNamesRow" select="1"/>
  <!-- В какой строке начинаются данные -->
  <xsl:param name="dataRowStart" select="2"/>
  <!-- Если false, то ячейки, не содержащие данных или содержащие -->
  <!-- только пробелы, пропускаются -->
  <xsl:param name="includeEmpty" select="true(  )"/>
  <!-- Если false, то не выводится комментарий, содержащий метаданные -->
  <!-- об авторе и дате создания-->
  <xsl:param name="includeComment" select="true(  )"/>

  <!-- Нормализовать namespacePrefix -->
  <xsl:variable name="nsp">
     <xsl:if test="$namespace">
        <!-- Использовать префикс только, если задано пространство имен -->
    <xsl:choose>
       <xsl:when test="contains($namespacePrefix,':')">
     <xsl:value-of
           select="concat(translate(substring-before(
                                      $namespacePrefix,
                                      ':'),' ',''),':')"/>
```

```
    </xsl:when>
    <xsl:when test="translate($namespacePrefix,' ','')">
       <xsl:value-of
          select="concat(translate($namespacePrefix,' ',''),':')"/>
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
    </xsl:if>
</xsl:variable>

<!-- Получить названия колонок, заменив в них пробелы  -->
<xsl:variable name="COLS" select="/*/*/*/ss:Row[$colNamesRow]/ss:Cell"/>

<xsl:template match="o:DocumentProperties">
  <xsl:if test="$includeComment">
     <xsl:text>&#xa;</xsl:text>
     <xsl:comment>
       <xsl:text>&#xa;</xsl:text>
          <xsl:if test="normalize-space(o:Company)">
               <xsl:text>Компания: </xsl:text>
               <xsl:value-of select="o:Company"/>
               <xsl:text>&#xa;</xsl:text>
          </xsl:if>
          <xsl:text>Автор: </xsl:text>
          <xsl:value-of select="o:Author"/>
          <xsl:text>&#xa;</xsl:text>
          <xsl:text>Дата создания: </xsl:text>
          <xsl:value-of select="translate(o:Created,'TZ',' ')"/>
          <xsl:text>&#xa;</xsl:text>
          <xsl:text>Последний автор: </xsl:text>
          <xsl:value-of select="o:LastAuthor"/>
          <xsl:text>&#xa;</xsl:text>
          <xsl:text>Дата сохранения:</xsl:text>
          <xsl:value-of select="translate(o:LastSaved,'TZ',' ')"/>
          <xsl:text>&#xa;</xsl:text>
     </xsl:comment>
  </xsl:if>
</xsl:template>

<xsl:template match="ss:Table">
  <xsl:element
       name="{concat($nsp,translate($topLevelName,
                    '&#x20;&#x9;&#xA;',$wsSub))}"
       namespace="{$namespace}">
```

```
    <xsl:apply-templates select="ss:Row[position(  ) >= $dataRowStart]"/>
  </xsl:element>
</xsl:template>

<xsl:template match="ss:Row">
  <xsl:element
       name="{concat($nsp,translate($rowName,
                    '&#x20;&#x9;&#xA;',$wsSub))}"
       namespace="{$namespace}">
    <xsl:for-each select="ss:Cell">
         <xsl:variable name="pos" select="position(  )"/>

       <!-- Получить правильное название колонки, даже если в исходной -->
       <!-- электронной таблице были пустые колонки -->
       <xsl:variable name="colName">
          <xsl:choose>
             <xsl:when test="@ss:Index and
                             $COLS[@ss:Index = current(  )/@ss:Index]">
            <xsl:value-of
              select="$COLS[@ss:Index = current(  )/@ss:Index]/ss:Data"/>
            </xsl:when>
            <xsl:when test="@ss:Index">
              <xsl:value-of
                select="$COLS[number(current(  )/@ss:Index)]/ss:Data"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="$COLS[$pos]/ss:Data"/>
            </xsl:otherwise>
          </xsl:choose>
       </xsl:variable>

       <xsl:if test="$includeEmpty or
                       translate(ss:Data,'&#x20;&#x9;&#xA;','')">
          <xsl:element
                name="{concat($nsp,translate($colName,
                                    '&#x20;&#x9;&#xA;',$wsSub))}"
                namespace="{$namespace}">
            <xsl:value-of select="ss:Data"/>
          </xsl:element>
       </xsl:if>

    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

```
  <xsl:template match="text(  )"/>

</xsl:stylesheet>
```

Если заданы параметры по умолчанию, то получается гораздо более естественное XML-представление:

```
<Table>
  <Row>
    <Date>20010817</Date>
    <Price>61.88</Price>
    <Volume>260163</Volume>
  </Row>
  <Row>
    <Date>20010820</Date>
    <Price>62.7</Price>
    <Volume>241859</Volume>
  </Row>
  <Row>
    <Date>20010821</Date>
    <Price>60.78</Price>
    <Volume>233989</Volume>
  </Row>
  <Row>
    <Date>20010822</Date>
    <Price>60.66</Price>
    <Volume>387444</Volume>
  </Row>
</Table>
```

XSLT 2.0

Улучшения за счет использования XSLT 2.0 сводятся в основном к нескольким вспомогательным функциям, которые позволяют убрать избыточный код, и к применению более лаконичных конструкций XPath 2.0.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/02/xpath-functions"
  xmlns:o="urn:schemas-microsoft-com:office:office"
  xmlns:x="urn:schemas-microsoft-com:office:excel"
  xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
  xmlns:ckbk="http://www.oreilly.com/xsltckbk">

  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- Имя элемента верхнего уровня -->
```

```
<xsl:param name="topLevelName" select=" 'Table' " as="xs:string"/>
<!-- Имя строки -->
<xsl:param name="rowName" select=" 'Row' " as="xs:string"/>
<!-- Название используемого пространства имен -->
<xsl:param name="namespace" select=" '' " as="xs:string"/>
<!-- Префикс используемого пространства имен -->
<xsl:param name="namespacePrefix" select=" '' " as="xs:string" />
<!-- Какой символ использовать вместо пробела в названиях колонок -->
<xsl:param name="wsSub" select="'_'" as="xs:string"/>
<!-- Какая строка содержит названия колонок -->
<xsl:param name="colNamesRow" select="1" as="xs:integer"/>
<!-- В какой строке начинаются данные -->
<xsl:param name="dataRowStart" select="2" as="xs:integer"/>
<!-- Если false, то ячейки, не содержащие данных или содержащие -->
<!-- только пробелы, пропускаются -->
<xsl:param name="includeEmpty" select="true( )" as="xs:boolean"/>
<!-- Если false, то не выводится комментарий, содержащий метаданные -->
<!-- об авторе и дате создания-->
<xsl:param name="includeComment" select="true( )" as="xs:boolean"/>

<!-- Нормализовать namespacePrefix -->
<xsl:variable name="nsp" as="xs:string"
  select="if (contains($namespacePrefix,':'))
   then concat(translate(substring-before($namespacePrefix,':'),' ',''),':')
   else
    if (matches($namespacePrefix,'\W'))
   then concat(translate($namespacePrefix,' ',''),':')
   else '' "/>

<!-- Получить названия всех колонок -->
<xsl:variable name="COLS" select="/*/*/*/ss:Row[$colNamesRow]/ss:Cell"/>

<xsl:template match="o:DocumentProperties">
  <xsl:if test="$includeComment">
     <xsl:text>&#xa;</xsl:text>
     <xsl:comment select="concat('&#xa;',
                              ckbk:comment(o:Company),
                              ckbk:comment(o:Author),
                              ckbk:comment(o:Created,'Created on'),
                              ckbk:comment(o:LastAuthor,'Last Author'),
                              ckbk:comment(o:LastSaved,'Saved on'))"/>
  </xsl:if>
  <xsl:text>&#xa;</xsl:text>
</xsl:template>
```

```
<xsl:template match="ss:Table">
  <xsl:element
       name="{ckbk:makeName($nsp,$topLevelName,$wsSub)}"
       namespace="{$namespace}">
     <xsl:apply-templates select="ss:Row[position() ge $dataRowStart]"/>
    </xsl:element>
</xsl:template>

<xsl:template match="ss:Row">
  <xsl:element
       name="{ckbk:makeName($nsp,$rowName,$wsSub)}"
       namespace="{$namespace}">
     <xsl:for-each select="ss:Cell">
       <xsl:variable name="pos" select="position()"/>

       <!-- Получить правильное название колонки, даже если в исходной -->
       <!-- электронной таблице были пустые колонки -->
       <xsl:variable name="colName" as="xs:string"
          select="if (@ss:Index and $COLS[@ss:Index = current()/@ss:Index])
                   then $COLS[@ss:Index = current()/@ss:Index]/ss:Datae
                   else
                   if (@ss:Index)
                   then $COLS[number(current()/@ss:Index)]/ss:Data
                   else $COLS[$pos]/ss:Data"/>

       <xsl:if test="$includeEmpty or
                      translate(ss:Data,'&#x20;&#x9;&#xA;','')">
       <xsl:element
            name="{ckbk:makeName($nsp,$colName,$wsSub)}"
            namespace="{$namespace}">
       <xsl:value-of select="ss:Data"/>
      </xsl:element>
     </xsl:if>
   </xsl:for-each>
  </xsl:element>
</xsl:template>

<xsl:template match="text()"/>

<xsl:function name="ckbk:makeName" as="xs:string">
  <xsl:param name="nsp" as="xs:string"/>
  <xsl:param name="name" as="xs:string"/>
  <xsl:param name="wsSub" as="xs:string"/>
  <xsl:sequence select="concat($nsp,translate($name,
```

```
                                    '&#x20;&#x9;&#xA;',$wsSub))"/>
  </xsl:function>

  <xsl:function name="ckbk:comment" as="xs:string">
    <xsl:param name="elem"/>
    <xsl:sequence select="ckbk:comment($elem, local-name($elem))"/>
  </xsl:function>

  <xsl:function name="ckbk:comment" as="xs:string">
    <xsl:param name="elem"/>
    <xsl:param name="label" as="xs:string"/>
    <xsl:sequence select="if (normalize-space($elem))
                          then concat($label,': ',$elem,'&#xa;')
                          else '' "/>
  </xsl:function>

</xsl:stylesheet>
```

Обсуждение

Я уже почти решил не включать этот рецепт в книгу, сочтя его тривиальным. Но потом понял, что для корректного решения нужно рассматривать ряд особых случаев, а во многих реализациях (включая и мою первую версию) это не делается. Например, часто в электронные таблицы включают пустые колонки в качестве разделителей. Как это обрабатывается, можете увидеть, поискав в тексте атрибут `@ss:Index`. Кроме того, в первой редакции этой книги в код были «зашиты» многие значения, которые теперь передаются параметрами.

Напрашивается по меньшей мере одно обобщение этой таблицы стилей – обработка случая, когда есть несколько элементов `ss:Worksheet`. Для этого можно было бы передать номер рабочего листа в качестве параметра.

```
  <xsl:param name="WSNum" select="1"/>

  <xsl:variable name="COLS"
       select="/*/ss:Worksheet[$WSNum]/*/ss:Row[$colNamesRow]/ss:Cell"/>

  <xsl:template match="ss:Workbook">
    <xsl:element name="{concat($nsp,translate($topLevelName,
                        '&#x20;&#x9;&#xA;',$wsSub))}"
                 namespace="{$namespace}">
      <xsl:apply-templates select="ss:Worksheet[number($WSNum)]/ss:Table"/>
    </xsl:element>
  </xsl:template>
```

Но интереснее было бы преобразовать каждый элемент `Worksheet` в документе с несколькими рабочими листами в отдельный элемент в результирующем

документе. При таком подходе названия колонок уже нельзя хранить в глобальной переменной:

```
<xsl:template match="ss:Workbook">
  <xsl:element name="{concat($nsp,translate($topLevelName,
                  '&#x20;&#x9;&#xA;',$wsSub))}"
              namespace="{$namespace}">
    <xsl:choose>
      <xsl:when test="number($WSNum) > 0">
        <xsl:apply-templates
          select="ss:Worksheet[number($WSNum)]/ss:Table">
        <xsl:with-param name="COLS"
            select="ss:Worksheet[number($WSNum)]
                                  /*/ss:Row[$colNamesRow]/ss:Cell"/>
        </xsl:apply-templates>
      </xsl:when>
      <xsl:otherwise>
        <xsl:for-each select="ss:Worksheet">
          <xsl:element
                 name="{concat($nsp,translate(@ss:Name,
                        '&#x20;&#x9;&#xA;',$wsSub))}"
                 namespace="{$namespace}">
    <xsl:apply-templates select="ss:Table">
      <xsl:with-param name="COLS"
                     select="*/ss:Row[$colNamesRow]/ss:Cell"/>
            </xsl:apply-templates>
          </xsl:element>
        </xsl:for-each>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:element>
</xsl:template>

<xsl:template match="ss:Table">
  <xsl:param name="COLS"/>
    <xsl:apply-templates select="ss:Row[position(  ) >= $dataRowStart]">
    <xsl:with-param name="COLS" select="$COLS"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="ss:Row">
  <xsl:param name="COLS"/>

  <!-- Все остальное не меняется ... -->

</xsl:template>
```

Единственный недостаток этого решения – предположение о том, что названия колонок в каждом рабочем листе должны находиться в одной и той же строке.

13.3. Генерация тематических карт из UML-модели с помощью XMI

Задача

Вы хотели бы воспользоваться своим любимым инструментом UML-моделирования с поддержкой XMI для создания описаний тематических карт на языке XTM.

Решение

Читателям, не знакомым с тематическими картами, возможно, стоит сначала прочитать раздел «Обсуждение».

Поскольку UML не задумывался для представления тематических карт, этот пример будет работать только при соблюдении определенных соглашений. По большей части, они касаются использования конкретных *стереотипов* UML. Стереотип – это механизм расширения UML, который позволяет ассоциировать с любым классификатором UML некий символ, который означает, что этому классификатору приписана определенная пользователем семантика. Для реализации тематических карт на UML нам потребуются стереотипы, перечисленные в таблице 13.2.

Таблица 13.2. *Соглашения об описании тематических карт на языке UML*

Стереотип	Контекст UML	Назначение
topic	Класс	Представляет тему. При наших соглашениях такая роль приписывается классу по умолчанию, поэтому этот стереотип не обязателен.
subject	Класс	Означает, что класс моделирует индикатор предмета (subject). Класс выступает в роли предмета, если на него ссылается некоторый элемент `subjectIndentityRef`. Точнее, этот стереотип позволяет отличить элементы `subjectIndicator`, ссылающиеся на индикатор опубликованного предмета, от тех, что ссылаются на тему или ресурс. См. *http://www.topicmaps.org/xtm/index.html#elt-subjectIdentity*.
resource	Класс	Означает, что класс представляет ресурс, используемый в качестве конечного элемента ассоциации тема-вхождение.
baseName	Класс	Означает, что класс моделирует альтернативное базовое имя темы и тем самым вводит новый контекст. Элементы контекста связываются с классом baseName посредством ассоциации со стереотипом *scope*.

Таблица 13.2. *Соглашения об описании тематических карт на языке UML (окончание)*

Стереотип	Контекст UML	Назначение
occurrence	Ассоциация	Означает, что ассоциация указывает на ресурс, который является вхождением темы.
scope	Ассоциация	Обозначает ассоциацию, определяющую контекст, в котором действуют характеристики тематической карты. Природа контекста зависит от стереотипа конечного класса (`topicRef`, `resourceRef` или `subjectIndicatorRef`).
variant	Ассоциация типа «обобщение» и класс	Соединяется с темой, вариантом которой является, посредством ассоциации типа «обобщение» со стереотипом variant. Параметры, управляющие применимостью варианта, кодируются в атрибутах класса. Сам класс также снабжается стереотипом variant, чтобы отличить его от темы.

Помимо этих стереотипов, используются следующие отображения UML на Topic Map:

- *Класс UML*
  Моделирует тему Topic Map, если только не задан стереотип, отличный от topic. В качестве базового имени темы используется имя класса. Если у класса есть атрибут ID. то в качестве идентификатора темы используется его значение по умолчанию. Если атрибута ID нет, то в качестве идентификатора темы выступает само имя класса.
- *Ассоциация UML*
  Моделирует ассоциацию Topic Map, если только не задан стереотип, обозначающий иное. Имя ассоциации UML отображается на имя ассоциации Topic Map. Спецификатор роли UML становится спецификатором роли Topic Map.
- *UML создает экземпляр ассоциации*
  Моделирует отношение instanceOf в Topic Map.
- *Простая ассоциация типа «обобщение» (наследование) в UML*
  Используется в качестве сокращенного способа задания канонической ассоциации superclass-subclass, в которой оконечным элементам автоматически назначаются роли superclass и subclass. То же самое отношение связывает классы со стереотипом baseName с классами-темами, для которых они представляют имя в альтернативном контексте. Обобщение со стереотипом variant определяет также варианты тем.

Если ваша версия Rational Rose не сохраняет модели в формате XMI, загрузите надстройку XMI или RoseXMI со страницы *http://www.rational.com/support/downloadcenter/addins/rose/ index.jsp*. Я тестировал этот пример только в Rose, но другие инструменты для работы с UML, например Together Control Center компании TogetherSoft, тоже поддерживают XMI, и для них пример, скорее всего, будет работать.

Ниже приведена таблица стилей для преобразования XMI в XTM. Я определил несколько XML-компонентов для удобства работы с длинными именами:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
    <!--= = = = = = = = = = = = = = = = = = = = =-->
    <!-- Конструкции для организации верхнего уровня XMI -->
    <!--= = = = = = = = = = = = = = = = = = = = =-->
    <!ENTITY FC "Foundation.Core">
    <!ENTITY FX "Foundation.Extension_Mechanisms">
    <!ENTITY FD "Foundation.Data_Types">
    <!ENTITY MM "Model_Management.Model">
    <!ENTITY ME "&FC;.ModelElement">

    <!--= = = = = = = = = = = = = = = = = = = = =-->
    <!-- Сокращения для основных элементов XMI, -->
    <!-- представляющих наибольший интерес для  -->
    <!-- этой таблицы стилей.                   -->
    <!--= = = = = = = = = = = = = = = = = = = = =-->
  <!-- Элемент UML общего вида -->
  <!ENTITY ELEM "&FC;.Namespace.ownedElement">
  <!-- Ассоциация как единое целое -->
  <!ENTITY ASSOC "&FC;.Association">
  <!-- Часть, представляющая соединение ассоциации -->
  <!ENTITY CONN "&ASSOC;.connection">
  <!-- Концы ассоциации. -->
  <!ENTITY END "&ASSOC;End">
  <!ENTITY CONNEND "&CONN;/&END;">
  <!ENTITY ENDTYPE "&END;.type">
  <!-- Класс UML -->
  <!ENTITY CLASS "&FC;.Class">
  <!-- Имя сущности UML -->
  <!ENTITY NAME "&FC;.ModelElement.name">
  <!-- Стереотип UML -->
  <!ENTITY STEREOTYPE "&ME;.stereotype/&FX;.Stereotype">
  <!-- Место, где сохраняется документация UML в формате XMI. -->
  <!-- Используется для данных о ресурсах -->
  <!ENTITY TAGGEDVALUE
     "&ME;.taggedValue/&FX;.TaggedValue/&FX;.TaggedValue.value">
  <!-- Отношение Supertype (наследование) -->
  <!ENTITY SUPERTYPE "&FC;.Generalization.supertype">
  <!ENTITY SUBTYPE "&FC;.Generalization.subtype">
  <!ENTITY SUPPLIER "&FC;.Dependency.supplier">
  <!ENTITY CLIENT "&FC;.Dependency.client">
  <!ENTITY DEPENDENCY
```

```
      "/XMI/XMI.content/&MM;/&ELEM;/&FC;.Dependency">
  <!ENTITY EXPRBODY
       "&FC;.Attribute.initialValue/&FD;.Expression/&FD;.Expression.body">
  <!ENTITY ATTR "&CLASS;ifier.feature/&FC;.Attribute">
  <!-- Используется для указания на стандартную спецификацию XTM -->
  <!ENTITY TM.ORG "http://www.topicmaps.org/xtm/index.html">
]>

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

<xsl:param name="termOnErr" select="true(  )"/>

   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
```

Воспользуемся ключами, чтобы упростить идентификацию стереотипов и обойти ассоциации UML, в которых встречаются перекрестные ссылки между атрибутами `xmi.id` и `xmi.idref`:

```
  <!-- Индексировать классы по имени -->
  <xsl:key name="classKey" match="&CLASS;" use="@xmi.id"/>
  <!-- Индексировать стереотипы по имени и xmi.id -->
  <xsl:key name="stereotypeKey"
                        match="&FX;.Stereotype" use="@xmi.id"/>
  <xsl:key name="stereotypeKey"
                        match="&FX;.Stereotype" use="&NAME;"/>

  <!-- Для кодирования тематических карт в UML применяются атрибуты xmi:id -->
  <!-- стереотипов. Мы воспользуемся ими в качестве эффективного средства -->
  <!-- проверки того, присоединен ли стереотип к элементу -->

  <xsl:variable name="OCCURENCE_ID"
                select="key('stereotypeKey','occurence')/@xmi.id"/>
  <xsl:variable name="RESOURCE_ID"
                select="key('stereotypeKey','resource')/@xmi.id"/>
  <xsl:variable name="TOPIC_ID"
                select="key('stereotypeKey','topic')/@xmi.id"/>
  <xsl:variable name="SUBJECT_ID"
                select="key('stereotypeKey','subject')/@xmi.id"/>
  <xsl:variable name="BASENAME_ID"
                select="key('stereotypeKey','baseName')/@xmi.id"/>
  <xsl:variable name="SCOPE_ID"
                select="key('stereotypeKey','scope')/@xmi.id"/>
```

```
<xsl:variable name="VARIANT_ID"
              select="key('stereotypeKey','variant')/@xmi.id"/>
```

Преобразовать UML-модель, записанную на языке XMI, в тематическую карту можно в два прохода. Сначала импортируем темы из классов, а затем – ассоциации XTM из ассоциаций UML:

```
<xsl:template match="/">
   <xtm:topicMap>
      <xsl:apply-templates mode="topics"/>
      <xsl:apply-templates mode="associations"/>
   </xtm:topicMap>
</xsl:template>
```

В темы следует транслировать только классы без стереотипа или со стереотипом `topic`. Все остальные встречающиеся в модели классы – это представления, на которые может ссылаться тема, например, индикаторы предмета и ресурсы:

```
<!--=  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =-->
<!-- Трансляция классов UML в темы  -->
<!--=  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =-->

<xsl:template match="&ELEM;/&CLASS;" mode="topics">
   <!-- Темы моделируются классами без стереотипа -->
   <!-- или со стереотипом 'topic'              -->
   <xsl:if test="not(&STEREOTYPE;/@xmi.idref) or
                      &STEREOTYPE;/@xmi.idref = $TOPIC_ID">
   <xsl:variable name="topicId">
      <xsl:call-template name="getTopicId">
         <xsl:with-param name="class" select="."/>
         <xsl:with-param name="prefix" select="''"/>
      </xsl:call-template>
   </xsl:variable>
   <xtm:topic id="{$topicId}">
      <!-- Этот цикл for-each нужен только для переключения контекста -->
      <!-- на необязательный атрибут Core.Attribute с именем
      'subjectIdentityid' -->
      <xsl:for-each select="&ATTR;[&NAME; = 'subjectIdentity']">
         <xtm:subjectIdentity>
            <xtm:subjectIdicatorRef xlink:href="{&EXPRBODY;}"/>
         </xtm:subjectIdentity>
      </xsl:for-each>
      <xtm:baseName>
         <xtm:baseNameString>
            <xsl:value-of select="&NAME;"/>
         </xtm:baseNameString>
```

```
      </xtm:baseName>
      <xsl:apply-templates select="." mode="getAlternateBaseNames"/>
      <xsl:apply-templates select="." mode="getVariants"/>
      <xsl:apply-templates select="." mode="getInstanceOf">
         <xsl:with-param name="classId" select="@xmi.id"/>
      </xsl:apply-templates>
      <xsl:apply-templates select="." mode="getOccurences"/>
    </xtm:topic>
  </xsl:if>
</xsl:template>

<!-- Вернуть идентификатор темы из класса темы; это -->
<!-- либо значение атрибута id, либо само имя класса -->
<xsl:template name="getTopicId">
   <xsl:param name="class"/>
   <xsl:param name="prefix" select="'#'"/>
   <xsl:for-each select="$class">
      <xsl:choose>
         <xsl:when test="&ATTR;/&NAME; = 'id' ">
            <xsl:value-of select="&ATTR;[&NAME; = 'id']/&EXPRBODY;"/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="concat($prefix,&NAME;)"/>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:for-each>
</xsl:template>

<!-- Вернуть идентификатор предмета из класса предмета; это -->
<!-- либо значение атрибута subjectIdentity, либо само имя класса -->
<xsl:template name="getSubjectIdentity">
   <xsl:param name="class"/>
   <xsl:for-each select="$class">
    <xsl:choose>
      <xsl:when test="&ATTR;/&NAME; = 'subjectIdentity' ">
         <xsl:value-of select="&ATTR;[&NAME; =
                              'subjectIdentity']&EXPRBODY;"/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="concat('#',&NAME;)"/>
      </xsl:otherwise>
   </xsl:choose>
   </xsl:for-each>
</xsl:template>
```

```
<!-- Вернуть идентификатор ресурса из класса ресурса; это -->
<!-- либо значение атрибута resourceName, либо само имя класса -->
<xsl:template name="getResourceIdentity">
   <xsl:param name="class"/>
   <xsl:for-each select="$class">
   <xsl:choose>
      <xsl:when test="&ATTR;/&NAME; = 'resourceName' ">
         <xsl:value-of select="&ATTR;[&NAME; =
                              'resourceName']/&EXPRBODY;"/>
         </xsl:when>
         <xsl:otherwise>
            <xsl:value-of select="concat('#',&NAME;)"/>
         </xsl:otherwise>
      </xsl:choose>
   </xsl:for-each>
</xsl:template>
```

Альтернативные базовые имена и варианты можно моделировать как специализации базового класса темы, пользуясь ассоциациями обобщения в UML. В зависимости от точки зрения это может показаться естественным делом или грубым осквернением идеи. Так или иначе, этот прием эффективен и на UML-диаграммах обозначается визуально, так что можно не полагаться на одни лишь метки стереотипов:

```
<!-- Альтернативные базовые имена находятся путем обхода -->
<!-- отношений обобщения и поиска стереотипов baseName -->

<xsl:template match="&ELEM;/&CLASS;" mode="getAlternateBaseNames">
  <xsl:variable name="xmiId" select="@xmi.id"/>
  <xsl:for-each select="../&FC;.Generalization
                 [&SUPERTYPE;/&CLASS;/@xmi.idref = $xmiId]">
    <xsl:variable name="subtypeXmiId"
             select="&FC;.Generalization.subtype/&CLASS;/@xmi.idref"/>
    <xsl:variable name="class" select="key('classKey',$subtypeXmiId)"/>
    <xsl:if test="$class/&STEREOTYPE;/@xmi.idref = $BASENAME_ID">
      <xsl:variable name="name" select="$class/&NAME;"/>
        <xtm:baseName>
          <xsl:call-template name="getScope">
            <xsl:with-param name="class" select="$class"/>
          </xsl:call-template>
          <xtm:baseNameString>
            <xsl:value-of select="substring-after($name,'::')"/>
          </xtm:baseNameString>
        </xtm:baseName>
  </xsl:if>
```

```
  </xsl:for-each>
</xsl:template>

<!-- Варианты находятся путем обхода отношений обобщения -->
<!-- и поиска стереотипов baseName -->

<xsl:template match="&ELEM;/&CLASS;" mode="getVariants">
  <xsl:variable name="xmiId" select="@xmi.id"/>
  <xsl:for-each select="../&FC;.Generalization
                        [&SUPERTYPE;/&CLASS;/@xmi.idref = $xmiId]">
    <xsl:variable name="subtypeXmiId"
         select="&FC;.Generalization.subtype/&CLASS;/@xmi.idref"/>
    <xsl:variable name="variantClass"
                  select="key('classKey',$subtypeXmiId)"/>
    <xsl:if test="$variantClass/&STEREOTYPE;/@xmi.idref = $VARIANT_ID">
      <xsl:variable name="name" select="$variantClass/&NAME;"/>
      <xtm:variant>
        <xtm:variantName>
          <xsl:call-template name="resourceRep">
            <xsl:with-param name="class" select="$variantClass"/>
          </xsl:call-template>
        </xtm:variantName>
        <xtm:parameters>
          <xsl:call-template name="getVariantParams">
            <xsl:with-param name="class" select="$variantClass"/>
          </xsl:call-template>
        </xtm:parameters>
        <!-- Сменить контекст на этот вариант, чтобы найти вложенные -->
        <!-- в него, если таковые существуют. -->
        <xsl:apply-templates select="$variantClass" mode="getVariants"/>
      </xtm:variant>
    </xsl:if>
  </xsl:for-each>
</xsl:template>

<!-- Получить параметры варианта из атрибутов класса варианта -->
<xsl:template name="getVariantParams">
  <xsl:param name="class"/>
  <xsl:if test="not($class/&ATTR;)">
    <xsl:message terminate="{$termOnErr}">
    У варианта должен быть хотя бы один параметр.
    </xsl:message>
  </xsl:if>
  <xsl:for-each select="$class/&ATTR;">
```

```
      <!-- Параметр моделируется как индикатор предмета -->
      <!-- или как ссылка на тему -->
      <xsl:choose>
        <xsl:when test="&STEREOTYPE;/@xmi.idref = $SUBJECT_ID">
          <xtm:subjectIdicatorRef xlink:href="{&EXPRBODY;}"/>
        </xsl:when>
        <xsl:otherwise>
          <xtm:topicRef xlink:href="{&EXPRBODY;}"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
  </xsl:template>
```

Вхождения в смысле Topic Map моделируются как ассоциации с классами, содержащими ссылки на ресурсы или данные. Поскольку встроенные данные ресурса могут оказаться слишком длинными для значения атрибута, то в этом примере мы разрешаем использовать описание атрибута как альтернативный контейнер для данных ресурса:

```
<!-- Вхождения в смысле Topic Map моделируются как ассоциации -->
  <!-- с классами, содержащими ссылки на ресурсы или данные -->
  <xsl:template match="&ELEM;/&CLASS;" mode="getOccurences">
    <xsl:variable name="xmiId" select="@xmi.id"/>
    <!-- Просмотреть все ассоциации, в которых участвует данный класс -->
    <xsl:for-each
          select="../&ASSOC;
                    [&CONN;/*/&ENDTYPE;/&CLASS;/@xmi.idref = $xmiId]">
      <!-- Проверить наличие стереотипа occurrence -->
      <xsl:if test="&STEREOTYPE;/@xmi.idref = $OCCURENCE_ID">
        <!-- Получить идентификатор ресурса, взглянув на другой конец -->
        <!-- ассоциации occurrence -->
        <xsl:variable name="resourceId"
                      select="&CONN;/*/&ENDTYPE;/&CLASS;
                                [@xmi.idref != $xmiId]/@xmi.idref"/>
        <!-- Получить класс, представляющий ресурс -->
        <xsl:variable name="resourceClass"
                      select="key('classKey',$resourceId)"/>
        <xtm:occurence>
        <xsl:apply-templates select="." mode="getInstanceOf">
          <xsl:with-param name="classId" select="$resourceId"/>
        </xsl:apply-templates>
        <!--TODO: Пока еще не умею это моделировать!
          <xsl:call-template name="getScope">
            <xsl:with-param name="class"/>
          </xsl:call-template>
```

```
-->
      <!-- Имеем либо ссылку на ресурс, либо данные ресурса. -->
      <!-- Если у класса есть атрибут resourceData, то второе -->
      <xsl:call-template name="resourceRep">
        <xsl:with-param name="class" select="$resourceClass"/>
      </xsl:call-template>
    </xtm:occurence>
  </xsl:if>
 </xsl:for-each>
</xsl:template>

<!-- Этот шаблон определяет, в каком виде представлен ресурс -->
<xsl:template name="resourceRep">
 <xsl:param name="class" />
 <xsl:variable name="resourceData">
   <!-- for-each для смены контекста -->
   <xsl:for-each select="$class/&ATTR;[&NAME; = 'resourceData']">
     <xsl:choose>
       <!-- Данные ресурса закодированы в документации -->
       <!-- к атрибуту UML -->
       <xsl:when test="&TAGGEDVALUE;">
         <xsl:value-of select="&TAGGEDVALUE;"/>
       </xsl:when>
       <!-- Данные ресурса закодированы в значении атрибута UML -->
       <xsl:otherwise>
         <xsl:value-of select="&EXPRBODY;"/>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:for-each>
  </xsl:variable>
  <!-- Если данные ресурса найдены, используем их. -->
  <!-- Иначе предполагаем, что пользователь имел в виду ссылку -->
  <xsl:choose>
   <xsl:when test="string($resourceData)">
     <xtm:resourceData>
       <xsl:value-of select="$resourceData"/>
     </xtm:resourceData>
   </xsl:when>
   <xsl:otherwise>
     <xsl:variable name="resource">
       <xsl:call-template name="getResourceIdentity">
         <xsl:with-param name="class" select="$class"/>
       </xsl:call-template>
     </xsl:variable>
```

```
      <xtm:resourceRef xlink:href="{$resource}"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Отношения XTM `instanceOf` моделируются как ассоциации *зависимости* в UML, которые также носят название *instantiates* (инстанцирует). Такое представление `instanceOf` выглядит вполне естественно:

```
<!-- Этот шаблон проверяет, есть ли для класса темы -->
<!-- какие-нибудь ассоциации instanceOf -->
<xsl:template match="&ELEM;/&CLASS;" mode="getInstanceOf">
  <xsl:param name="classId"/>
  <!-- В цикле обходим все отношения зависимости и -->
  <!-- смотрим, как представлен экземпляр -->
  <xsl:for-each
      select="&DEPENDENCY;[&CLIENT;/&CLASS;/@xmi.idref = $classId]">
    <xtm:instanceOf>
      <xsl:variable name="instanceClass"
            select="key('classKey',&SUPPLIER;/&CLASS;/@xmi.idref)"/>
      <!-- Выясняем, как моделируется экземпляр: как предмет или как
      тема -->
      <xsl:variable name="sterotypeId"
                  select="$instanceClass/&STEREOTYPE;/@xmi.idref"/>
      <xsl:choose>
        <!-- Это случай индикатора предмета -->
        <xsl:when test="$sterotypeId = $SUBJECT_ID">
          <xsl:variable name="subjectIdentity">
            <xsl:call-template name="getSubjectIdentity">
              <xsl:with-param name="class" select="$instanceClass"/>
            </xsl:call-template>
          </xsl:variable>
          <xsl:if test="not(normalize-space($subjectIdentity))">
            <xsl:message terminate="{$termOnErr}">
            Предмет без идентификатора!
            </xsl:message>
          </xsl:if>
          <xtm:subjectIdicatorRef xlink:href="{$subjectIdentity}"/>
        </xsl:when>
        <!-- В противном случае экземпляр представлен темой -->
        <xsl:when test="not($sterotypeId) or $sterotypeId = $TOPIC_ID">
          <xsl:variable name="topicId">
            <xsl:call-template name="getTopicId">
              <xsl:with-param name="class" select="$instanceClass"/>
            </xsl:call-template>
```

```
            </xsl:variable>
            <xsl:if test="not(normalize-space($topicId))">
              <xsl:message terminate="{$termOnErr}">
              Тема без идентификатора!
              </xsl:message>
            </xsl:if>
            <topicRef xlink:href="{$topicId}"/>
          </xsl:when>
           <xsl:otherwise>
            <xsl:message terminate="{$termOnErr}">
              <xsl:text>instanceOf должна указывать на тему или
              предмет. </xsl:text>
              <xsl:value-of select="$instanceClass/&NAME;"/>
              <xsl:text> is a </xsl:text>
              <xsl:value-of
                  select="key('stereotypeKey',$sterotypeId)/&NAME;"/>
              <xsl:text>.&#xa;</xsl:text>
            </xsl:message>
           </xsl:otherwise>
        </xsl:choose>
      </xtm:instanceOf>
    </xsl:for-each>
</xsl:template>

<xsl:template name="getScope">
    <xsl:param name="class"/>
    <xsl:variable name="classesAssociations"
                  select="/*/XMI.content/*/&ELEM;
                          /&ASSOC;
                          [&CONN;/*/
                          &FC;.AssociationEnd.type/
                          &CLASS;/@xmi.idref = $class/@xmi.id]"/>
    <xsl:variable name="scopeAssociations"
                 select="$classesAssociations[
                          &FC;.ModelElement.stereotype/
                          &FX;.Stereotype/
                          @xmi.idref = $SCOPE_ID]"/>
    <xsl:if test="$scopeAssociations">
      <xtm:scope>
         <xsl:for-each select="$scopeAssociations">
         <xsl:variable name="targetClassId"
            select="&CONN;/*/&ENDTYPE;/&CLASS;
                           [@xmi.idref != $class/@xmi.id]/@xmi.idref"/>
      <xsl:variable name="targetClass"
```

```
                        select="key('classKey',$targetClassId)"/>
        <xsl:call-template name="getScopeRef">
          <xsl:with-param name="class" select="$targetClass"/>
        </xsl:call-template>
         </xsl:for-each>
      </xtm:scope>
    </xsl:if>
  </xsl:template>

  <xsl:template name="getScopeRef">
    <xsl:param name="class"/>
    <xsl:variable name="stereotypeId"
                  select="$class/&FC;.ModelElement.stereotype/
                          FX;.Stereotype/
                          @xmi.idref"/>
    <xsl:choose>
      <xsl:when test="not($stereotypeId) or $stereotypeId = $TOPIC_ID">
        <xsl:variable name="topidId">
          <xsl:call-template name="getTopicId">
            <xsl:with-param name="class" select="$class"/>
          </xsl:call-template>
        </xsl:variable>
        <xtm:topicRef xlink:href="{$topidId}"/>
      </xsl:when>
      <xsl:when test="$stereotypeId = $SUBJECT_ID">
        <xsl:variable name="subjectId">
          <xsl:call-template name="getSubjectIdentity">
            <xsl:with-param name="class" select="$class"/>
          </xsl:call-template>
        </xsl:variable>
        <xtm:subjectIndicatorRef xlink:href="{$subjectId}"/>
      </xsl:when>
      <xsl:when test="$stereotypeId = $RESOURCE_ID">
        <xsl:variable name="resourceId">
          <xsl:call-template name="getResourceIdentity">
            <xsl:with-param name="class" select="$class"/>
          </xsl:call-template>
        </xsl:variable>
        <xtm:resourceRef xlink:href="{$resourceId}"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:message terminate="{$termOnErr}">
        Контекстом может быть topicRef, subjectRef или resourceRef!
        </xsl:message>
```

```
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="text(  )" mode="topics"/>
```

Следующие далее шаблоны – часть второго прохода, на котором ассоциации UML, которые еще не были обработаны в силу наличия специальных стереотипов, преобразуются в ассоциации Topic Map. Здесь стереотипом ассоциации является элемент `topicRef`, который определяет вид моделируемой ассоциации. Это, конечно, профанация самой идеи стереотипа, но что делать, если UML не предоставляет никакого другого подходящего места для хранения этой информации. Ассоциации в контексте – это проблема, которую я решил игнорировать. Со всех остальных точек зрения ассоциации UML хорошо ложатся на концепцию тематических карт:

```
<!--=  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =-->
<!-- АССОЦИАЦИИ UML В АССОЦИАЦИИ TOPIC MAP -->
<!--=  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =-->
<xsl:template match="&ASSOC;" mode="associations">
   <!-- Только именованные ассоциации UML могут быть ассоциациями
   Topic Map-->
   <xsl:if test="normalize-space(&NAME;)">
      <xtm:association id="{&NAME;}">
        <xtm:instanceOf>
           <topicRef
              xlink:href="{key('stereotypeKey',
                              &STEREOTYPE;/@xmi.idref)/&NAME;}"/>
           </xtm:instanceOf>
           <xsl:for-each select="&CONNEND;">
              <xtm:member>
                 <xtm:roleSpec>
                     <xtm:topicRef xlink:href="{&NAME;}"/>
                 </xtm:roleSpec>
                 <xsl:variable name="topicId">
                     <xsl:call-template name="getTopicId">
                       <xsl:with-param name="class"
                          select="key('classKey',
                             &ENDTYPE;/&CLASS;/@xmi.idref)"/>
                     </xsl:call-template>
                 </xsl:variable>
                 <xtm:topicRef xlink:href="{$topicId}"/>
              </xtm:member>
           </xsl:for-each>
        </xtm:association>
```

```
    </xsl:if>
</xsl:template>

<xsl:template match="&ELEM;/&FC;.Generalization"
              mode="associations">

<xsl:variable name="subClassId"
              select="&SUBTYPE;/&CLASS;/@xmi.idref"/>
<xsl:variable name="subClass"
              select="key('classKey',$subClassId)"/>
<xsl:variable name="superClassId"
              select="&SUPERTYPE;/&CLASS;/@xmi.idref"/>
<xsl:variable name="superClass"
              select="key('classKey',$superClassId)"/>
```

Если между двумя темами существует отношение обобщения, то используем его как признак канонического отношения superclass-subclass. В спецификации XTM это важное отношение поддержано явно с помощью публикуемых индикаторов предметов (published subject indicators – PSI). В идеале следовало бы опираться на отсутствие стереотипа у обобщения, но в той версии XMI, с которой я работал, информации о стереотипах для обобщений вообще нет:

```
<xsl:if test="(not($subClass/&STEREOTYPE;/@xmi.idref) or
               $subClass/&STEREOTYPE;/@xmi.idref = $TOPIC_ID) and
                    (not($superClass/&STEREOTYPE;/@xmi.idref) or
                     $superClass/&STEREOTYPE;/@xmi.idref = $TOPIC_ID)">

  <xtm:association>
    <xsl:variable name="id">
      <xsl:choose>
        <xsl:when test="normalize-space(&NAME;)">
          <xsl:value-of select="&NAME;"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="@xmi.id"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>

    <xsl:attribute name="id">
      <xsl:value-of select="$id"/>
    </xsl:attribute>

    <xtm:instanceOf>
      <subjectIndicatorRef
```

```
            xlink:href="&TM.ORG;#psi-superclass-subclass"/>
        </xtm:instanceOf>

        <xtm:member>

          <xtm:roleSpec>
            <xtm:subjectIndicatorRef xlink:href="&TM.ORG;#psi-superclass"/>
          </xtm:roleSpec>

          <xsl:variable name="superClassTopicId">
            <xsl:call-template name="getTopicId">
              <xsl:with-param name="class" select="$superClass"/>
            </xsl:call-template>
          </xsl:variable>
          <xtm:topicRef xlink:href="{$superClassTopicId}"/>

        </xtm:member>

        <xtm:member>
          <xtm:roleSpec>
              <xtm:subjectIndicatorRef xlink:href="&TM.ORG;#psi-subclass"/>
              </xtm:roleSpec>

              <xsl:variable name="subClassTopicId">
                 <xsl:call-template name="getTopicId">
                    <xsl:with-param name="class" select="$subClass"/>
                 </xsl:call-template>
              </xsl:variable>

              <xtm:topicRef xlink:href="{$subClassTopicId}"/>
            </xtm:member>

         </xtm:association>
      </xsl:if>
   </xsl:template>

   <xsl:template match="text(  )" mode="associations"/>

</xsl:stylesheet>
```

Обсуждение

С помощью тематических карт представляются знания о предметах реального мира. Это позволяет людям и компьютерам находить нужную информацию

точнее и быстрее. Впервые о тематических картах заговорили в 1993 году, когда идея была опубликована в рабочем документе Davenport Group[1]. Затем эта парадигма была расширена в контексте исследований института GCA Research Institute (ныне называется IDEAlliance) в связи с приложениями стандарта HyTime (*http://www.hytime.org/papers/htguide.html*). Спецификация языка XTM была разработана как побочный продукт этой работы под контролем независимой организации TopicMaps.org.

Тема (topic) – это электронный аналог предмета (субъекта, subject) реального мира. Оззи Осборн – субъект реального мира, но, поскольку создать реального Оззи в компьютере не получится, то в качестве суррогата мы создаем тему «Оззи». У тем есть имена, которые называются *базовыми именами*. У каждой темы может быть одно универсальное имя (правильно называть его *естественным* (unconstrained)) и несколько других, употребляемых в определенных контекстах (scope). *Контекстом* называется совокупность условий, в которых характеристики тематической карты имеют силу. В нашем примере это может означать, что тема «Оззи Осборн» в юридическом контексте то же самое, что John Michael Osbourne[2].

Тема может указывать на *вхождение*; так называется ресурс, который предоставляет относящуюся к теме информацию. Ресурс может ссылаться на адресуемое содержимое (`resourceRef`) или сам являться таким содержимым (`resourceData`).

Тема может принимать участие в специальной ассоциации `instanceOf`, которая говорит о том, что данная тема является экземпляром более общего класса объектов. Такой класс можно обозначить ссылкой на другую тему или на индикатор предмета. *Индикаторы предметов* – интересная особенность тематических карт. Они предлагают способ описания природы предмета посредством ассоциирования его со стандартным публикуемым адресом, который может, например, устанавливаться государственной организацией.

Темы могут быть связаны ассоциациями. *Ассоциация* – это именованное отношение между двумя или более темами, причем каждая тема играет в ассоциации определенную роль.

Ряд других механизмов, присущих тематических картам, позволяют моделировать знания о предметах. Интересующихся читателей отсылаем к спецификации XTM по адресу *http://www.TopicMaps.org*. В сравнении с другими спецификациями эта написана в расчете на человека, совсем незнакомого с предметом. Почти вся функциональность XTM отображается на те или иные конструкции UML (как описано в разделе «Решение»). Наибольшую трудность для отображения представляет понятие контекста. В парадигме тематических карт *контекст* – это способ сообщить, что некоторую характеристику темы следует принимать во внимание только в определенных обстоятельствах. Контекст применим к базовым именам, ассоциациям и вхождениям. Хотя сформулированные в решении соглашения

---

[1] Группа Davenport Group основана компанией Unix System и другими организациями, включая издательский дом O'Reilly & Associates.

[2] Если только Джон Майкл с соблюдением всех юридических процедур не поменял имя на Оззи; тогда этот контекст представляет интерес только для его матери.

позволяют учесть контекст для базовых имен, сделать это для ассоциаций и вхождений пока не удается. Связано это с тем, что соглашения подразумевают моделирование в виде UML-ассоциаций, а в UML ассоциации контекстно независимы. Можно было бы смоделировать этот механизм посредством ограничений UML. Но к сожалению та версия XMI, которая реализована в программе Rational Rose, не включает информацию об ограничениях. На практике контекст вообще редко бывает нужен, так что эта проблема, возможно, и не станет помехой, особенно для начинающих пользователей и составителей тематических карт.

На рис. 13.2 приведен пример тематической карты, представленной в нотации UML. Эта тема моделирует информацию о томатах в контексте еды. Пример, конечно, шуточный, но я взял его потому, что Сэм Хантинг приводит такой же пример в своей книге *XML Topic Maps: Creating and Using Topic Maps for the Web* (Addison Wesley, 2002). На нем он продемонстрировал различные свойства тематических карт, что позволяет проверить точность получившегося в результате XMI-документа.

Ниже приведен фрагмент получившегося XTM-файла:

```
<xtm:topicMap xmlns:xtm='http://www.topicmaps.org/xtm/1.0' xmlns:xlink=
"http://www.w3.org/1999/xlink">
  <xtm:topic id="EN">
    <xtm:subjectIdentity>
      <xtm:subjectIdicatorRef
        xlink:href="http://www.topicmaps.org/xtm/1.0/language.xtm#en"/>
    </xtm:subjectIdentity>
    <xtm:baseName>
      <xtm:baseNameString>EN</xtm:baseNameString>
    </xtm:baseName>
  </xtm:topic>
  <xtm:topic id="FR">
    <xtm:subjectIdentity>
      <xtm:subjectIdicatorRef
        xlink:href="http://www.topicmaps.org/xtm/1.0/language.xtm#fr"/>
    </xtm:subjectIdentity>
    <xtm:baseName>
      <xtm:baseNameString>FR</xtm:baseNameString>
    </xtm:baseName>
  </xtm:topic>
  <xtm:topic id="myTomato">
    <xtm:subjectIdentity>
      <xtm:subjectIdicatorRef
          xlink:href="http://www.fed.gov/usda/doc/tomato.htm#gradeA"/>
      </xtm:subjectIdentity>
      <xtm:baseName>
        <xtm:baseNameString>tomato</xtm:baseNameString>
      </xtm:baseName>
      <xtm:baseName>
```

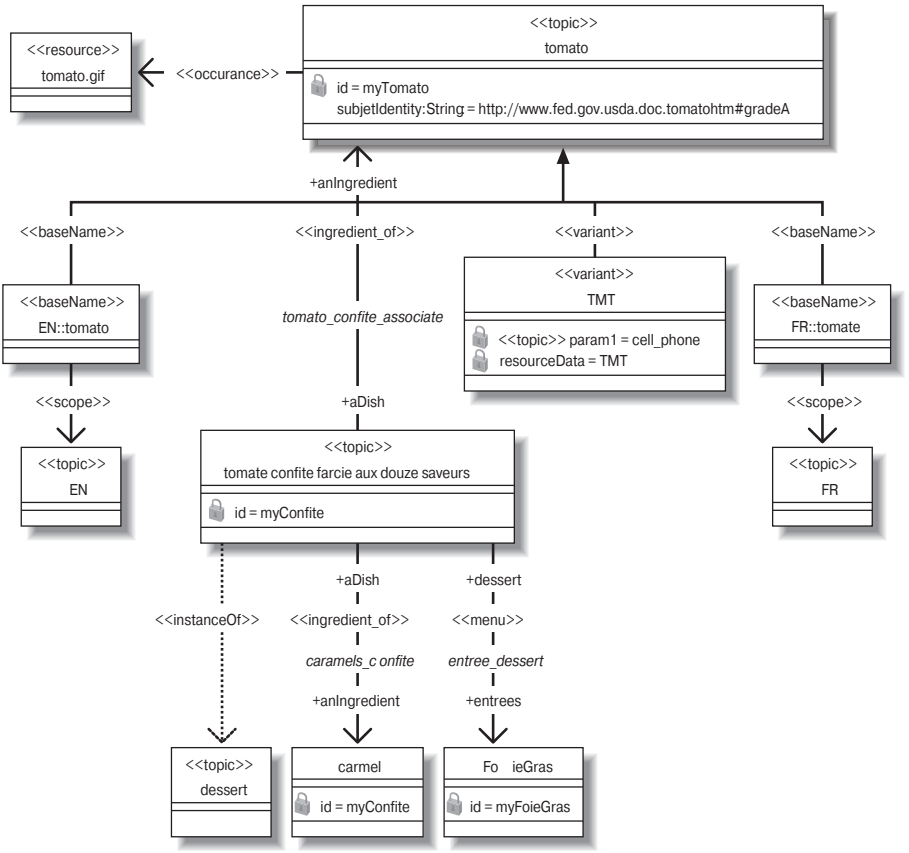


Рис. 13.2. UML-диаграмма тематической карты томатов

```
      <xtm:scope>
         <xtm:topicRef xlink:href="#EN"/>
      </xtm:scope>
      <xtm:baseNameString>tomato</xtm:baseNameString>
   </xtm:baseName>
   <xtm:baseName>
      <xtm:scope>
         <xtm:topicRef xlink:href="#FR"/>
      </xtm:scope>
      <xtm:baseNameString>tomate</xtm:baseNameString>
   </xtm:baseName>
   <xtm:variant>
      <xtm:variantName>
         <xtm:resourceData>TMT</xtm:resourceData>
      </xtm:variantName>
      <xtm:parameters>
```

```
            <xtm:topicRef xlink:href="cell_phone"/>
            <xtm:topicRef xlink:href="TMT"/>
         </xtm:parameters>
      </xtm:variant>
      <xtm:occurence>
         <xtm:resourceRef xlink:href="#tomato.gif"/>
      </xtm:occurence>
   </xtm:topic>
   <xtm:topic id="myConfite">
      <xtm:baseName>
         <xtm:baseNameString>tomate confite farcie aux douze saveurs
         </xtm:baseNameString>
      </xtm:baseName>
      <xtm:instanceOf>
         <topicRef xlink:href="#desert"/>
      </xtm:instanceOf>
   </xtm:topic>

<!-- Опущено -->

   <xtm:association id="tomato_confite_association">
      <xtm:instanceOf>
         <topicRef xlink:href="ingredient_of"/>
      </xtm:instanceOf>
      <xtm:member>
         <xtm:roleSpec>
            <xtm:topicRef xlink:href="anIngredient"/>
         </xtm:roleSpec>
         <xtm:topicRef xlink:href="myTomato"/>
      </xtm:member>
      <xtm:member>
         <xtm:roleSpec>
            <xtm:topicRef xlink:href="aDish"/>
         </xtm:roleSpec>
         <xtm:topicRef xlink:href="myConfite"/>
      </xtm:member>
   </xtm:association>
   <xtm:association id="caramels_confite">
      <xtm:instanceOf>
         <topicRef xlink:href="ingredient_of"/>
      </xtm:instanceOf>
      <xtm:member>
         <xtm:roleSpec>
            <xtm:topicRef xlink:href="anIngredient"/>
```

```
            </xtm:roleSpec>
            <xtm:topicRef xlink:href="myCarmel"/>
         </xtm:member>
         <xtm:member>
            <xtm:roleSpec>
               <xtm:topicRef xlink:href="aDish"/>
            </xtm:roleSpec>
            <xtm:topicRef xlink:href="myConfite"/>
         </xtm:member>
      </xtm:association>
<!-- Опущено -->
</xtm:topicMap>
```

См. также

Стандарты UML и XMI разработаны организацией Object Management Group (OMG). Дополнительную информацию см. на сайтах *http://www.omg.org/uml/* и *http://www.omg.org/technology/xml/index.htm*.

Официальным сайтом, посвященным тематическим картам и языку XTM, является TopicMaps.org (*http://www.topicmaps.org*).

В рецепте 13.5 объясняется, как из тематических карт можно сгенерировать Web-сайт.

13.4. Генерация Web-сайтов из тематических карт

Задача

Требуется получить знания о предмете, представленном в тематической карте, и на этой основе сгенерировать с помощью XSLT Web-сайт.

Решение

В основу решения положен каркас Cogitative Topic Maps Web Site (CTW), представленный публике в сборнике статей *XML Topic Maps* под редакцией Джека Парка (Jack Park) (Addison Wesley, 2002). Оригинальная работа была выполнена в компании Extreme Markup Languages в 2000 году.

В каркасе CTW принято следующее отображение элементов тематических карт на HTML:

Элемент тематической карты	HTML-разметка
Тематическая карта	Web-сайт
Тема	Web-страница
Тематические ассоциации	Карта сайта
Вхождения тем	Рисунки, графика, текст, фрагменты HTML и т.д.
Имена тем	Заголовки страниц, списки и текст гиперссылок.

Предмет нашей тематической карты – алгоритмы, точнее, алгоритмы сортировки. Представленные в карте знания собраны из различных информационных источников в Интернете и организованы в виде ассоциаций типа класс-подкласс между алгоритмами и вхождениями описаний типов, демонстраций и примеров кода на нескольких языках программирования.

Определившись с предметом и содержанием Web-сайта, мы можем естественно выстроить онтологию субъектов и объектов. Онтологический уровень в CTW состоит из двух основных частей: классификация предметов темы сайта и классификация характеристик темы, составляющих содержание страниц.

Обе классификации играют важную роль в управлении внешним обликом сайта. Типы тем определяют верстку страницы, а типы характеристик тем – стили отдельных элементов и строительных блоков. Результаты изображены на рис. 13.3.

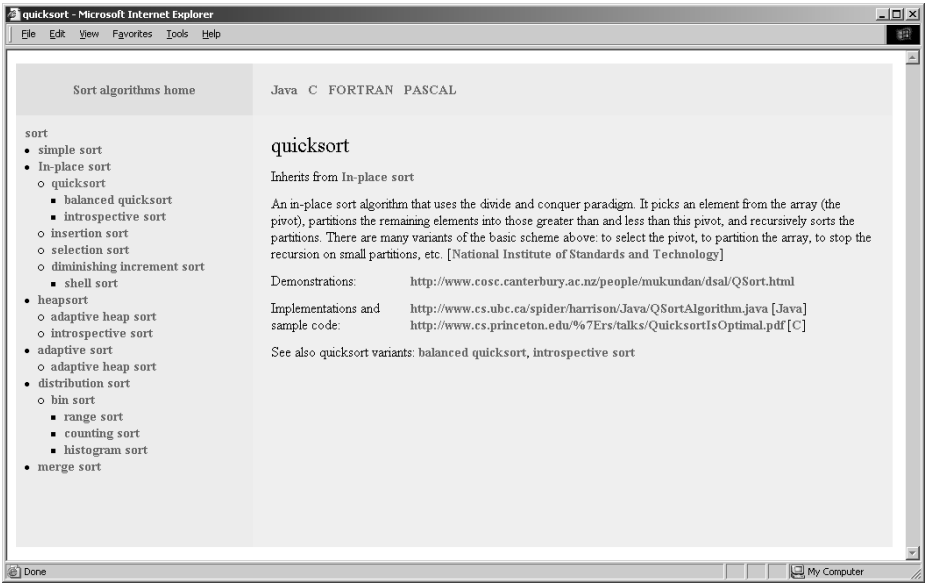


Рис. 13.3. Внешний вид сгенерированного сайта

В следующих разделах описываются предметы, представленные на сайте «Алгоритмы сортировки».

Алгоритмы сортировки

Основные предметы нашего сайта – это различные подклассы класса «алгоритм сортировки».

```
<topic id="sort">
   ...
</topic>
<association>
```

```
    <instanceOf>
       <topicRef xlink:href="#_class-subclass"/>
    </instanceOf>
    <member>
       <roleSpec>
          <topicRef xlink:href="#_superclass"/>
       </roleSpec>
       <topicRef xlink:href="#sort"/>
    </member>
    <member>
       <roleSpec>
          <topicRef xlink:href="#_subclass"/>
       </roleSpec>
       <topicRef xlink:href="#simplesort"/>
       <topicRef xlink:href="#in-place sort"/>
       <topicRef xlink:href="#heapsort"/>
       <topicRef xlink:href="#adaptivesort"/>
       <topicRef xlink:href="#distributionsort"/>
       <topicRef xlink:href="#mergesort"/>
    </member>
</association>
```

Полная версия тематической карты и XSLT-сценариев есть на сайте *http://www.cogx.com/ctw*. В этом разделе мы приводим только некоторые фрагменты для иллюстрации различных команд.

У любого алгоритма сортировки могут быть свои подклассы, например, можно насчитать четыре разновидности *сортировки на месте* (`in-place-sort`), у каждого из которых в свою очередь могут быть подклассы:

```
<association>
   <instanceOf>
      <topicRef xlink:href="#_class-subclass"/>
   </instanceOf>
   <member>
      <roleSpec>
         <topicRef xlink:href="#_superclass"/>
      </roleSpec>
      <topicRef xlink:href="#in-place sort"/>
   </member>
   <member>
      <roleSpec>
         <topicRef xlink:href="#_subclass"/>
      </roleSpec>
      <topicRef xlink:href="#quicksort"/>
```

```
      <topicRef xlink:href="#insertionsort"/>
      <topicRef xlink:href="#selsort"/>
      <topicRef xlink:href="#dimincrsort"/>
    </member>
  </association>
```

Национальный институт стандартов и технологий США (National Institute of Standards and Technology (NIST)) поддерживает замечательный сайт, посвященный алгоритмам (*http://www.nist.gov/dads/*), на котором каждому алгоритму отведена отдельная страница. В нашей карте URL этих страниц служат идентификаторами предметов:

```
<topic id="insertionsort">
  <subjectIdentity>
    <subjectIndicatorRef
      xlink:href="http://www.nist.gov/dads/HTML/insertsrt.html"/>
  </subjectIdentity>
```

Помимо отношения к другим алгоритмам, которое описывается ассоциациями класс-подкласс, у алгоритмов сортировки есть и иные характеристики, например, базовое имя и вхождения.

В данной тематической карте алгоритмам сортировки присвоены имена, под которыми они широко известны:

```
  <baseName>
    <baseNameString>сортировка вставками</baseNameString>
  </baseName>
```

Иногда имеются альтернативные имена, представленные как базовые имена в контексте `also-known-as`:

```
  <baseName>
    <scope>
      <topicRef xlink:href="#also-known-as"/>
    </scope>
    <baseNameString>линейная сортировка вставками
    </baseNameString>
  </baseName>
```

Описание алгоритма представлено в виде вхождения типа `description` в контексте источника описания. Следующий код содержит цитату с сайта Национального института стандартов и технологий США (по этой причине она приведена в контексте темы `nist`). Прочесть его можно так: «В контексте NIST сортировка вставками описывается как ...».

```
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#description"/>
    </instanceOf>
    <scope>
```

```
        <topicRef xlink:href="#nist"/>
      </scope>
      <resourceData>Выбирается следующий элемент и вставляется
        в конечную структуру данных в нужное место относительно
        вставленных ранее элементов. </resourceData>
  </occurrence>
```

Ссылки на демонстрации работы алгоритмов, например аплеты и анимации, представлены вхождениями типа `demonstration`:

```
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#demo"/>
    </instanceOf>
    <resourceRef xlink:href=
    "http://www.cosc.canterbury.ac.nz/people/mukundan/dsal/ISort.html"/>
  </occurrence>
```

Ссылки на реализации алгоритмов сортировки тоже можно представить в карте в виде вхождений типа `code sample`, описанных в контексте языка программирования. Языки программирования – это другой класс тем, представленных на сайте, с собственной независимой навигацией.

```
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#code"/>
    </instanceOf>
    <scope>
      <topicRef xlink:href="#fortran"/>
    </scope>
    <resourceRef
    xlink:href="http://gams.nist.gov/serve.cgi/Module/TOMS/505/8547"/>
  </occurrence>
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#code"/>
    </instanceOf>
    <scope>
      <topicRef xlink:href="#java"/>
    </scope>
    <resourceRef xlink:href=
    "http://www.cs.ubc.ca/spider/harrison/Java/InsertionSortAlgorithm.java"/>
  </occurrence>
</topic>
```

Это вся информация об алгоритмах, которую мы решили включить. Далее мы построим на ее основе заголовки страниц, ссылки на родственные алгоритмы,

описания, ссылки на Web-страницы, где можно найти определение алгоритма, ссылки на демонстрацию работы алгоритма и примеры кода со ссылками на языки программирования, на которых эти примеры реализованы.

Языки программирования

Нас интересует лишь тот факт, что языки программирования с точки зрения названия и определения являются экземплярами класса `programming language`:

```
<topic id="java">
  <subjectIdentity>
    <subjectIndicatorRef
      xlink:href="http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=java"/>
  </subjectIdentity>
  <instanceOf>
    <topicRef xlink:href="#plang"/>
  </instanceOf>
  <baseName>
    <baseNameString>Java</baseNameString>
  </baseName>
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#definition"/>
    </instanceOf>
    <scope>
      <topicRef xlink:href="#cnet"/>
    </scope>
    <resourceData>Язык Java был разработан компанией Sun Microsystems
    для добавления анимации и других действий к Web-сайтам. Небольшие
    приложения (аплеты), созданные на Java могут выполняться в любой
    графической системе, имеющей доступ к Web, но для этого необходим
    браузер с поддержкой Java. Согласно описанию Sun Java — "простой
    объектно-ориентированный, распределенный, интерпретируемый,
    надежный, безопасный, архитектурно-независимый, переносимый,
    высокопроизводительный, многопоточный, динамический, современный,
    универсальный язык программирования". </resourceData>
  </occurrence>
</topic>
```

Показанная на рис. 13.4 страница описания языка программирования содержит ссылки на примеры кода, написанного на этом языке, и перекрестные ссылки на реализованные алгоритмы.

Корневая тема

В каркасе CTW `root` – это тема, для которой индикатором предмета служит сам документ, содержащий тематическую карту. В терминологии тематических

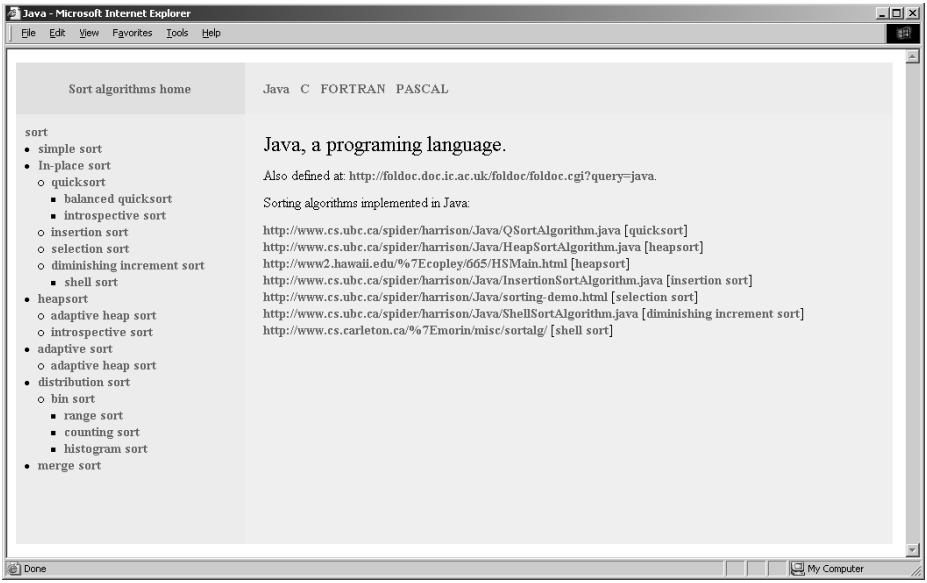


Рис. 13.4. Страница описания языка программирования

карт корневая тема – материализация того документа, которому она принадлежит:

```
<topic id="default">
    <subjectIdentity>
        <subjectIndicatorRef xlink:href="#map"/>
    </subjectIdentity>
```

При объединении тематических карт в CTW эта тема добавляется в контексты всех характеристик тем. Важнее, впрочем, тот факт, что она соответствует начальной странице сайта.

В следующем примере имя корневой темы отображается в виде гиперссылки в левом верхнем углу на любой странице сайта:

```
<baseName>
   <baseNameString>Алгоритмы сортировки</baseNameString>
</baseName>
```

Здесь самое место поместить аннотации к тематической карте. Мы ограничились лишь описанием проекта и копирайтом.

```
<occurrence>
   <instanceOf>
   <topicRef xlink:href="#definition"/>
   </instanceOf>
   <resourceData><![CDATA[ Этот сайт посвящен алгоритмам, точнее,
   алгоритмам сортировки. <br><br>
```

```
      Он создан для рецепта, описывающего каркас CTW в книге XSLT
      Cookbook издательства O'Reilly.]]> </resourceData>
  </occurrence>
</topic>
```

На начальной странице (рис. 13.5) сайта показано только описание проекта.

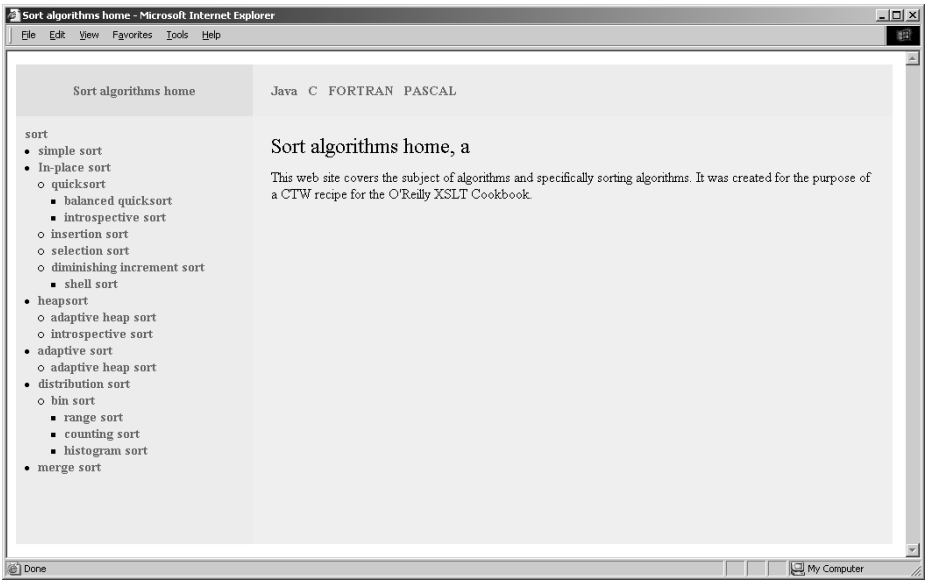


Рис. 13.5. Начальная страница сайта, посвященного алгоритмам сортировки

Элементы и верстка страницы

Сначала определим переменную `root`, которая будет представлять корневую тему. Как было сказано выше, индикатором этой темы является документ, который содержит тематическую карту:

```
<xsl:variable name="root"
    select="//topic[subjectIdentity/subjectIndicatorRef/@xlink:href
                                    = concat('#',/topicMap/@id)]"/>
```

Тот же узел можно было бы найти с помощью ключа `subjectIndicator`, который ищет темы по адресам индицирующих их ресурсов:

```
<xsl:key
    name = "subjectIndicator"
    match = "topic"
    use = "subjectIdentity/subjectIndicatorRef/@xlink:href" />

<xsl:variable name="root"
            select="key('subjectIndicator',concat('#',/topicMap/@id))"/>
```

Первым делом генерируем начальную страницу, вызывая шаблон верстки `root-page` для корневой темы:

```
<xsl:template match="/">
    <xsl:call-template name="root-page">
           <xsl:with-param name="this" select="$root"/>
    </xsl:call-template>
```

Затем генерируем страницы для каждого подкласса класса `sorting algorithm`. Здесь вызывает шаблон верстки `algorithm-page`, которому в качестве параметра передается строка `sorting algorithm`. Шаблон рекурсивно вызывает себя самого для обхода всех подклассов до самого нижнего уровня:

```
  <xsl:call-template name="algorithm-page">
    <xsl:with-param name="this" select="key('topicByID','#sort')"/>
  </xsl:call-template>
```

Таблица стилей генерирует страницы для каждого языка программирования, вызывая шаблон `plang-page`:

```
  <xsl:for-each select="key('instanceOf','#plang')">
    <xsl:call-template name="plang-page"/>
  </xsl:for-each>
</xsl:template>
```

Ключ `instanceOf` возвращает все экземпляры класса с заданным идентификатором темы класса:

```
<xsl:key
    name = "instanceOf"
    match = "topic"
    use = "instanceOf/topicRef/@xlink:href" />
```

Ключ `topicByID` возвращает все элементы `topic` по заданному идентификатору темы:

```
<xsl:key
    name = "instanceOf"
    match = "topic"
    use = "instanceOf/topicRef/@xlink:href" />
```

На рисунках выше видно, что все три шаблона верстки разбивают страницу на четыре прямоугольных области. Этим управляет общий главный шаблон страницы. Сначала мы просим процессор создать выходной файл относительно папки, заданной параметром `$out-dir` и формируем заголовок TITLE, помещая в него базовое имя текущей темы в естественном контексте. Последнее достигается сопоставлением с элементами `topic` в режиме `label`. Затем начинается процедура разбиения страницы на четыре части. В левом верхнем углу создается гиперссылка на начальную страницу, ее текст – имя корневой темы в естественном контексте. Эту

задачу решает применение шаблона к элементам `topic` в режиме `link`. В правом верхнем углу находятся ссылки на страницы языков программирования. Чтобы создать их, мы обходим все экземпляры темы `programming language`. В левом нижнем углу страницы под ссылкой на начальную страницу выводится часть карты сайта, соответствующая классификации алгоритмов сортировки. Наконец, в основной части страницы (правый нижний угол) выводится содержимое, относящееся к текущей теме, которая была передана в шаблон `page` в качестве параметра `content`:

```
<xsl:template name="page">
  <xsl:param name="this"/>
  <xsl:param name="content"/>
  <redirect:write select="concat($out-dir,$this/@id,'.html')">
  <HTML>
     <HEAD>
    <TITLE>
       <xsl:apply-templates select="$this" mode="label"/>
    </TITLE>
     </HEAD>
    <BODY>
    <table width="1000" height="100%" cellspacing="0" cellpadding="10">
    <tr>
    <td width="250" height="20" bgcolor="#ffddbb" align="center">
       <xsl:apply-templates select="$root" mode="link"/>
    </td>
    <td width="750" height="20" valign="top" bgcolor="#eeeeee">
       <table cellspacing="10">
       <tr>
          <xsl:for-each select="key('instanceOf','#plang')">
           <td background="grey">
              <xsl:apply-templates select="." mode="link"/>
           </td>
         </xsl:for-each>
       </tr>
        </table>
      </td>
    </tr>
    <tr>
      <td valign="top" bgcolor="#eeeeee">
        <xsl:call-template name="sitemap">
          <xsl:with-param name="classRef">#sort</xsl:with-param>
          <xsl:with-param name="current" select="$this/@id"/>
        </xsl:call-template>
      </td>
      <td valign="top" bgcolor="#ffeedd" >
```

```
            <xsl:copy-of select="$content"/>
          </td>
        </tr></table>
        </BODY>
    </HTML>
    </redirect:write>
</xsl:template>
```

Вот как шаблон применяется к элементам `topic` в режиме `label`:

```
<xsl:template match="topic" mode="label">
    <xsl:value-of select="baseName[not(scope)]/baseNameString"/>
</xsl:template>
```

А в режиме `link` элемент `baseName` используется для формирования гиперссылки на страницу темы:

```
<xsl:template match="topic" mode="link">
  <a href="{@id}.html">
     <xsl:value-of select="baseName[not(scope)]/baseNameString"/>
  </a>
</xsl:template>
```

Ниже нам встретится также сопоставление в режиме `subject-indicator`, когда элемент `baseName` применяется для формирования гиперссылки на ресурс, индицирующий сопоставленную тему:

```
<xsl:template match="topic" mode="indicator">
    <a href="{subjectIdentity/subjectIndicatorRef/@xlink:href}">
       <xsl:value-of select="baseName[not(scope)]/baseNameString"/>
    </a>
</xsl:template>
```

Шаблон `sitemap` обходит все подклассы темы `sort`, создает гиперссылки на страницы, соответствующие производным алгоритмам, и рекурсивно вызывает себя для обработки следующих в иерархии подклассов:

```
<xsl:template name="sitemap">
  <xsl:param name="classRef"/>
  <xsl:param name="current"/>
  <xsl:variable name="topic" select="key('topicByID',$classRef)"/>
  <xsl:choose>
     <xsl:when test="$topic/@id=$current">
        <span class="A">
           <xsl:apply-templates select="$topic" mode="label"/>
        </span>
     </xsl:when>
     <xsl:otherwise>
```

```
        <xsl:apply-templates select="$topic" mode="link"/>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:variable name="aref" select="key('classAssoc',$classRef)"/>
    <xsl:if test="$aref">
      <ul>
        <xsl:for-each
             select="$aref/member
                     [roleSpec/topicRef/@xlink:href=
                      '#_subclass']/topicRef">
          <li>
          <xsl:call-template name="sitemap">
            <xsl:with-param name="classRef" select="@xlink:href"/>
            <xsl:with-param name="current" select="$current"/>
          </xsl:call-template>
          </li>
      </xsl:for-each>
      </ul>
    </xsl:if>
</xsl:template>
```

Ключ `classRef`, применяемый в этом шаблоне, использует идентификатор заданной темы для нахождения ассоциаций типа `superclass-subclass`, в которых эта тема играет роль суперкласса:

```
<xsl:key
    name = "classAssoc"
    match = "association[instanceOf/topicRef/@xlink:href=
                         '#_class-subclass']"
    use = "member[roleSpec/topicRef/@xlink:href=
                  '#_superclass']/topicRef/@xlink:href" />
```

Далее нам встретится еще ключ `subClassRef`, который также индексирует ассоциации `superclass-subclass`, но на этот раз возвращает элемент `member`, для которого заданная тема выступает в роли подкласса:

```
<xsl:key
    name = "subClassAssoc"
    match = "association[instanceOf/topicRef/@xlink:href=
                         '#_class-subclass']"
    use = "member[roleSpec/topicRef/@xlink:href=
                  '#_subclass']/topicRef/@xlink:href" />
```

Теперь мы готовы к рассмотрению трех шаблонов верстки.

Шаблон `root-page` совсем простой. Он вызывает описанный ранее шаблон `page`, передавая ему сгенерированную HTML-разметку для вхождения типа `description` в параметре `content`:

```
<xsl:template name="root-page">
  <xsl:param name="this"/>
  <xsl:call-template name="page">
     <xsl:with-param name="this" select="$this"/>
     <xsl:with-param name="content">
        <font size="+1">
           <xsl:apply-templates
              select="$this/occurrence
                 [instanceOf/topicRef/@xlink:href='#description']"/>
        </font>
     </xsl:with-param>
  </xsl:call-template>
</xsl:template>
```

Шаблон `plang-page` чуть сложнее. Он формирует заголовок, составленный из имени текущей темы, за которой следует название ее типа. Далее идет строка, содержащая ссылку на то место в Web, где определен предмет темы. Затем выводятся описания темы, если таковые имеются. Ищутся и выводятся ссылки на все вхождения кода на текущем языке. После ресурса в квадратных скобках размещается ссылка на алгоритм сортировки, реализованный в этом ресурсе:

```
<xsl:template name="plang-page">
   <xsl:param name="this" select="."/>
   <xsl:call-template name="page">
      <xsl:with-param name="this" select="$this"/>
      <xsl:with-param name="content">
         <font size="+2">
            <xsl:apply-templates select="$this" mode="label"/>, a
            <xsl:apply-templates mode="label"
          select="key('topicByID',$this/instanceOf/topicRef/@xlink:href)" />.
         </font>
         <br/><br/>
         <xsl:apply-templates select="$this/subjectIdentity"/>
         <xsl:apply-templates
            select="$this/occurrence
                   [instanceOf/topicRef/@xlink:href='#description']"/>
         <xsl:variable name="codes"
            select="key('plang-codes',concat('#',$this/@id))"/>
         <xsl:if test="$codes">
            <span>Алгоритмы сортировки, реализованные на языке
               <xsl:apply-templates select="$this" mode="label"/>:</span>
            <ul>
               <xsl:for-each select="$codes">
                  <li>
```

```
                    <a href="{resourceRef/@xlink:href}">
                        <xsl:value-of select="resourceRef/@xlink:href"/>
                    </a>
                    [<xsl:apply-templates select=".." mode="link"/>]<br/>
                  </li>
                </xsl:for-each>
              </ul>
              <br/><br/>
            </xsl:if>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:template>
```

Ключ `plang-codes` индексирует все вхождения в тематической карте по теме в том же контексте:

```
<xsl:key
    name="plang-codes"
    match="occurrence"
    use="scope/topicRef/@xlink:href"/>
```

Последним идет шаблон `algorithm-page`. Его заголовок состоит из имени текущей темы, за которым в квадратных скобках следуют имена тем, перечисленных в элементах `also-known-as`. За строкой идентификации предмета следует список суперклассов, которым наследует текущий алгоритм сортировки, если таковые имеются. За ним расположены одно или несколько описаний алгоритма, список ссылок на вхождения демонстраций работы алгоритма и список примеров кода с перекрестными ссылками на соответствующие им языки программирования в квадратных скобках. В конце страницы располагается список подклассов или вариантов текущего алгоритма, если таковые имеются. И наконец шаблон `algorithm-page` вызывает себя самого для вывода страниц, описывающих подклассы текущего класса:

```
<xsl:template name="algorithm-page">
  <xsl:param name="this"/>
  <xsl:call-template name="page">
    <xsl:with-param name="this" select="$this"/>
    <xsl:with-param name="content">
      <font size="+2"><xsl:apply-templates select="$this" mode="label"/>
        <xsl:if test="$this/baseName
                    [scope/topicRef/@xlink:href='#also-known-as']">
      [<xsl:value-of select="$this/baseName
          [scope/topicRef/@xlink:href='#also-known-as']
           /baseNameString"/>]
        </xsl:if>
      </font>
```

```
  <br/><br/>
  <xsl:apply-templates select="$this/subjectIdentity"/>
  <xsl:variable name="superclasses"
    select="key('subClassAssoc',concat('#',$this/@id))
    member[roleSpec/topicRef/@xlink:href='#_superclass']/topicRef"/>
  <xsl:if test="$superclasses">
    Inherits from
    <xsl:for-each select="$superclasses">
      <xsl:apply-templates
        select="key('topicByID',@xlink:href)" mode="link"/>
        <xsl:if test="position( ) != last( )">, </xsl:if>
    </xsl:for-each>
    <br/><br/>
  </xsl:if>
  <xsl:apply-templates
    select="$this/occurrence
                [instanceOf/topicRef/@xlink:href='#description']"/>
  <xsl:variable name="demos"
    select="$this/occurrence
                [instanceOf/topicRef/@xlink:href='#demo']"/>
  <xsl:if test="$demos">
    <span>Демонстрации: </span>
    <ul>
      <xsl:for-each select="$demos">
        <li>
          <a href="{resourceRef/@xlink:href}"><
          xsl:value-of select="resourceRef/@xlink:href"/>
        </a><br/>
      </li>
    </xsl:for-each>
  </ul>
  <br/>
</xsl:if>
<xsl:variable name="codes"
  select="$this/occurrence[instanceOf/topicRef/@xlink:href='#code']"/>
<xsl:if test="$codes">
  <span>Реализации и примеры кода: </span>
  <ul>
    <xsl:for-each select="$codes">
      <li>
        <a href="{resourceRef/@xlink:href}">
          <xsl:value-of select="resourceRef/@xlink:href"/>
        </a>
        [<xsl:apply-templates mode="link"
```

```
            select="key('topicByID',scope/topicRef/@xlink:href)"/>]
        </li>
      </xsl:for-each>
    </ul>
    <br/>
  </xsl:if>
  <xsl:variable name="subclasses"
    select="key('classAssoc',
                concat('#',$this/@id))/member
                                        [roleSpec/topicRef/@xlink:href=
                                        '#_subclass']/topicRef"/>
    <xsl:if test="$subclasses">
      See also
      <xsl:value-of select="$this/baseName[not(scope)]/baseNameString"/>
      variants:
      <xsl:for-each select="$subclasses">
        <xsl:apply-templates
            select="key('topicByID',@xlink:href)" mode="link"/>
        <xsl:if test="position(  ) != last(  )">, </xsl:if>
       </xsl:for-each>
     </xsl:if>
    </xsl:with-param>
  </xsl:call-template>
  <xsl:variable name="aref"
     select="key('classAssoc',concat('#',$this/@id))"/>
  <xsl:for-each
        select="$aref/member
              [roleSpec/topicRef/@xlink:href='#_subclass']/topicRef">
        <xsl:call-template name="algorithm-page">
          <xsl:with-param name="this" select="key('topicByID',@xlink:href)"/>
      </xsl:call-template>
   </xsl:for-each>
</xsl:template>
```

Обсуждение

Topic Maps – это технология сбора и управления знаниями о различных предметных областях. В данном случае мы представили отношения между алгоритмами и другими объектами и ресурсами. Если скрупулезно следовать соглашениям, принятым в каркасе CTW, то из исходной тематической карты можно создать Web-сайт.

В приведенном решении мы ограничились небольшим количеством объектов и отношений между ними. Реальные приложения гораздо сложнее. Мы лишь ставили себе целью продемонстрировать огромные возможности, заложенные в каркасе CTW.

В каркасе CTW единственный документ, содержащий тематическую карту, управляет как содержанием, так и структурой целого сайта. Если карта организована правильно, то обеспечивается надежная и интуитивно понятная процедура поддержания ссылок между страницами, содержимого страниц и метаданных. Сайты, построенные с помощью каркаса CTW, легко объединяются и не страдают от проблемы «битых ссылок». Применение XSLT гарантирует единообразный внешний вид, независимость от платформы и возможность повторного использования.

За все эти удобства приходится платить: каркас CTW заставляет вас осмысливать содержимое в терминах тем, их характеристик и ассоциаций. Вы можете создавать содержимое только в рамках выбранной онтологии, но зато знания, представленные на сайте, будут хорошо организованы, а навигация по ним удобна.

В основу CTW положен язык XSLT, потому что он обеспечивает модульные и удобные для сопровождения средства преобразования и стилизации знаний, содержащихся в тематической карте. Динамические решения на базе CTW и XSLT масштабируются до нескольких тысяч тем, а статические ограничены только располагаемым дисковым пространством.

См. также

XML Topic Maps: Creating and Using Topic Maps for the Web, под редакцией Jack Park, (Addison Wesley, 2002) – прекрасная книга, в которой в доступной форме рассматривается теория и практика применения тематических карт.

Хорошую подборку ссылок на ресурсы, посвященные тематическим картам, можно найти по адресу *http://www.dmoz.org/Computers/Artificial_Intelligence / Knowledge_Representation/Topic_Maps/*.

Автор поддерживает сайт *http://www.cogx.com/ctw*, где также есть материалы о тематических картах и каркасе CTW. Слайды с презентации CTW на конференции Extreme Markup Languages находятся по адресу *http://www.cogx.com/Extreme2000/*.

13.5. Генерация документации о SOAP из WSDL-документа

Задача

Вы создаете архитектуру предприятия на основе Web-сервисов с использованием SOAP и WSDL. Требуется, чтобы разработчики могли найти полную и полезную информацию об имеющихся сервисах.

Решение

В этом решении мы создадим на основе WSDL-описания сервер документации по Web-сервисам, то есть сервис, который предоставляет информацию о сервисах[1]. Мы напишем CGI-сценарий на языке Perl, который будет с помощью

[1] Если хотите, можете назвать его метасервисом.

XSLT обрабатывать один WSDL-файл. Этот файл содержит сам или включает информацию обо всех сервисах, работающих на предприятии. В данном случае CGI-сценарий вызывает процессор XSLT с помощью функции `system`. Такой способ еще годится для создания прототипа, но никак не для промышленного развертывания. Было бы лучше воспользоваться Perl-модулями `XML::LibXML` и `XML::LibXSLT`. А еще лучше прибегнуть к развитой серверной подсистеме на базе XSLT, например, Cocoon. Мы приняли упрощенный подход для того, чтобы сосредоточиться на тех аспектах, которые относятся к XSLT и WSDL, а не к архитектуре CGI.

Главная страница сайта генерируется CGI-сценарием, которые показывает имеющиеся сервисы и порты. О том, что это такое, см. ниже. Мы следующим образом вызываем процессор Saxon из Perl:

```
#!c:/perl/bin/perl
print "Content-type: text/html\n\n" ;
system "saxon StockServices.wsdl wsdlServiceList.xslt" ;
```

Следующее преобразование строит форму, в которой описаны сервисы, порты, привязки и типы портов:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  exclude-result-prefixes="wsdl soap http mime">

  <xsl:output method="html"/>

  <xsl:template match="/">
     <html>
     <head>
        <title>Сервисы в компании ACME Web Services, Inc.</title>
        <xsl:comment>Документация из WSDL: сгенерирована
           wsdlServiceList.xslt
        </xsl:comment>
     </head>
     <body>
        <xsl:apply-templates/>
     </body>
     </html>

  </xsl:template>
```

```
<xsl:template match="wsdl:definitions">
   <h1>Сервисы в компании ACME Web Services, Inc.</h1>
   <br/>
   <form name="QueryServiceForm" method="post" action="QueryService.pl">
   <table>
     <tbody>
      <tr>
         <td>Сервисы</td>
         <td>
            <select name="service">
               <option value="ALL">BCE</option>
               <xsl:apply-templates select="wsdl:service"/>
            </select>
         </td>
      </tr>
      <tr>
         <td>Порты</td>
         <td>
            <select name="port">
               <option value="ALL">BCE</option>
               <xsl:apply-templates select="wsdl:service/wsdl:port"/>
            </select>
         </td>
      </tr>
      <tr>
         <td>Привязки</td>
         <td>
            <select name="binding">
               <option value="ALL">BCE</option>
               <xsl:apply-templates select="wsdl:binding"/>
            </select>
      </td>
     </tr>
     <tr>
       <td>Типы портов</td>
       <td>
          <select name="portType">
             <option value="ALL">BCE</option>
             <xsl:apply-templates select="wsdl:portType"/>
          </select>
       </td>
     </tr>
   </tbody>
</table>
```

```
    <br/>
    <button type="submit" name="submit">Послать запрос</button>
    </form>
</xsl:template>

<xsl:template match="wsdl:service | wsdl:port | wsdl:binding | wsdl:portType">
  <option value="{@name}"><xsl:value-of select="@name"/></option>
</xsl:template>

</xsl:stylesheet>
```

Пользователь может выбрать любую комбинацию сервиса, порта, привязки и типа порта, о которой хочет получить подробную информацию. Для обработки запроса вызывается небольшой Perl-сценарий, который разбирает входные данные и вызывает еще одно XSLT-преобразование:

```
#!/perl/bin/perl

use warnings;

use strict;
use CGI qw(:standard);

my $query = new CGI ;

my $service = $query->param('service');
my $port = $query->param('port');
my $binding = $query->param('binding');
my $portType = $query->param('portType');

print $query->header('text/html');

system "saxon StockServices.wsdl QueryService.xslt service=$service port=$port
binding=$binding portType=$portType"
```

Это преобразование находит сервисы, соответствующие заданному пользователем критерию, и отображает подробную информацию о каждом из них. В первой части таблицы стилей некоторые параметры приводятся к каноническому виду. Делать это необходимо, потому что атрибуты `@element` в WSDL-файле могут содержать префикс пространства имен, а атрибут `@name` его не содержит. То же самое относится и к конструкции `key`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xslt [
  <!ENTITY TNSPREFIX "'acme:'">
]>
<xsl:stylesheet version="1.0"
```

```
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/">

<!-- Параметры запроса -->
<xsl:param name="service" select="'ALL'"/>
<xsl:param name="port" select="'ALL'"/>
<xsl:param name="binding" select="'ALL'"/>
<xsl:param name="portType" select="'ALL'"/>

<!-- Способ (или трюк), позволяющий присвоить -->
<!-- переменной пустое значение, если соответствующий -->
<!-- параметр равен 'ВСЕ', а в противном случае записать -->
<!-- в нее результат конкатенации &TNSPREFIX и параметра. -->
<!-- Например, number($service = 'ВСЕ') * 99999 + 1 -->
<!-- равно 1, если $service не равно 'ВСЕ', и равно 100000, -->
<!-- если равно. Число 100000 заведомо больше длины строки, -->
<!-- поэтому substring вернет пустую строку. Все это нужно -->
<!-- только для того, чтобы упростить формирование -->
<!-- перекрестных ссылок -->
<xsl:variable name="serviceRef"
              select="substring(concat(&TNSPREFIX;,$service),
                                number($service = 'ALL') * 99999 + 1)"/>
<xsl:variable name="portRef"
              select="substring(concat(&TNSPREFIX;,$port),
                                number($port = 'ALL') * 99999 + 1)"/>
<xsl:variable name="bindingRef"
              select="substring(concat(&TNSPREFIX;,$binding),
                                number($binding = 'ALL') * 99999 + 1)"/>
<xsl:variable name="portTypeRef"
              select="substring(concat(&TNSPREFIX;,$portType),
                                number($portType = 'ALL') * 99999 + 1)"/>

<!-- Эти ключи упрощают и ускоряют поиск -->
<xsl:keyname="bindings_key"
        match="wsdl:binding" use="concat(&TNSPREFIX;,@name)"/>
<xsl:keyname="portType_key"
        match="wsdl:portType" use="concat(&TNSPREFIX;,@name)"/>

<xsl:output method="html"/>

  <xsl:template match="/">
```

```
      <html>
    <head>
       <title>ACME Web Services, Inc. Результат запроса</title>
      <xsl:comment>Документация из WSDL: сгенерирована wsdlServiceList.xslt
       </xsl:comment>
    </head>
    <body>
       <xsl:apply-templates select="wsdl:definitions"/>
    </body>
     </html>
  </xsl:template>
```

Далее, чтобы легче было понять, соответствует ли запросу хотя бы один сервис, мы запоминаем результат в переменной и проверяем ее нормализованное значение. Если оно пусто, значит подходящих сервисов не нашлось, и мы выводим соответствующее сообщение. Запрос довольно хитрый, поскольку для того чтобы узнать тип порта сервиса, нужно сначала перейти от сервиса к привязке, а потом получить тип порта для этой привязки:

```
<xsl:template match="wsdl:definitions">
   <xsl:variable name="result">
     <!-- Найти сервисы, соответствующие параметрам запроса. -->
     <!-- Для этого необходимо перейти от порта сервиса к -->
     <!-- привязке, а затем к типу привязки. Отсюда и -->
     <!-- вложенные вызовы key(). -->
     <xsl:apply-templates
           select="wsdl:service[
                  (not($serviceRef)  or @name = $service) and
                  (not($portRef)  or wsdl:port/@name = $port) and
                  (not($bindingRef) or wsdl:port/@binding = $bindingRef)
                  and
                  (not($portTypeRef) or key('portType_key',
                                            key('bindings_key',
                                              wsdl:port/@binding)/@type)
                                               /@name = $portType)]"/>
    </xsl:variable>
    <xsl:choose>
     <xsl:when test="normalize-space($result)">
        <xsl:copy-of select="$result"/>
     </xsl:when>
     <xsl:otherwise>
        <p><b>Подходящих сервисов не найдено</b></p>
     </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Оставшаяся часть таблицы стилей – это преимущественно преобразование WSDL в HTML-таблицу. У сервиса может быть несколько портов, поэтому выбрать порты, соответствующие параметрам запроса, придется еще раз:

```
<xsl:template match="wsdl:service" mode="display">
<h1><xsl:value-of select="@name"/></h1>
<p><xsl:value-of select="wsdl:documentation"/></p>
<table border="1" cellpadding="5" cellspacing="0"  width="600">
   <tbody>
      <xsl:apply-templates
         select="wsdl:port[(not($portRef) or @name = $port) and
                          (not($bindingRef) or @binding = $bindingRef)]"
         mode="display"/>
   </tbody>
</table>
</xsl:template>
```

Используем ключи для перехода от порта к привязке (операция соединения) и от привязки к типу порта:

```
<xsl:template match="wsdl:port" mode="display">
   <tr>
      <td style="font-weight:bold" colspan="2" align="center">Порт</td>
   </tr>
   <tr>
      <td colspan="2"><h3><xsl:value-of select="@name"/></h3></td>
   </tr>
   <tr>
      <td style="font-weight:bold" width="50">Привязка</td>
      <td><xsl:value-of select="substring-after(@binding,':')"/></td>
   </tr>
   <tr>
      <td style="font-weight:bold" width="50">Адрес</td>
      <td><xsl:value-of select="soap:address/@location"/></td>
   </tr>
   <tr>
      <th colspan="2" align="center">Операции</th>
   </tr>
   <xsl:apply-templates select="key('bindings_key',@binding)" mode="display"/>
</xsl:template>


<xsl:template match="wsdl:binding" mode="display">
   <xsl:apply-templates select="key('portType_key',@type)" mode="display">
      <xsl:with-param name="operation" select="wsdl:operation/@name"/>
   </xsl:apply-templates>
```

```
</xsl:template>

<xsl:template match="wsdl:portType" mode="display">
   <xsl:param name="operation"/>
   <xsl:for-each select="wsdl:operation[@name = $operation]">
      <tr>
         <td colspan="2"><h3><xsl:value-of select="@name"/></h3></td>
      </tr>
      <xsl:if test="wsdl:input">
         <tr>
            <td style="font-weight:bold" width="50">Вход</td>
            <td><xsl:value-of select="substring-after(wsdl:input/
               @message,':')"/></td>
         </tr>
         <xsl:variable name="msgName"
            select="substring-after(wsdl:input/@message,':')"/>
         <xsl:apply-templates
            select="/*/wsdl:message[@name = $msgName]" mode="display"/>
   </xsl:if>
      <xsl:if test="wsdl:output">
         <tr>
            <td style="font-weight:bold" width="50">Выход</td>
            <td><xsl:value-of select="substring-after(wsdl:output/
               @message,':')"/></td>
         </tr>
         <xsl:variable name="msgName"
            select="substring-after(wsdl:output/@message,':')"/>
         <xsl:apply-templates
            select="/*/wsdl:message[@name = $msgName]" mode="display"/>
      </xsl:if>
   </xsl:for-each>
</xsl:template>

<xsl:template match="wsdl:message" mode="display">
   <xsl:variable name="dataType"
        select="substring-after(wsdl:part[@name='body']/@element,':')"/>
   <tr>
      <td colspan="2">
         <xsl:apply-templates
              select="/*/wsdl:types/*/*[@name=$dataType]" mode="display">
            <xsl:with-param name="initial-newline" select="false(  )"/>
         </xsl:apply-templates>
     </td>
```

```
    </tr>
</xsl:template>
```

Следующий код выводит XSD-схему сообщений в виде XML внутри HTML. Сама схема – это, пожалуй, самая полезная информация для документирования типов данных, ассоциированных с входными и выходными сообщениями сервиса:

```
<xsl:template match="*" mode="display">
  <xsl:param name="initial-newline" select="true(  )"/>

  <xsl:if test="$initial-newline">
    <xsl:call-template name="newline"/>
  </xsl:if>
  <!-- открывающий тег -->
  <a>&lt;</a>
  <a><xsl:value-of select="name(.)" /> </a>

  <!-- Выводим атрибуты -->
  <xsl:for-each select="@*">
    <a><xsl:text> </xsl:text><xsl:value-of select="name(.)" />  </a>
    <a>=&quot;</a>
    <xsl:value-of select="." />
    <a>&quot;</a>
  </xsl:for-each>

  <xsl:choose>
    <xsl:when test="child::node(  )">
      <!-- закрыть тег -->
      <a>&gt;</a>
      <xsl:apply-templates  mode="display"/>
      <xsl:call-template name="newline"/>
      <!-- закрывающий тег -->
      <a>&lt;</a>
      <a>/<xsl:value-of select="name(.)" /></a>
    </xsl:when>
    <xsl:otherwise>
      <a>/</a>
    </xsl:otherwise>
  </xsl:choose>
  <a>&gt;</a>
</xsl:template>

<!-- Начать новую строку и сформировать отступ, вычисляемый по уровню -->
<!-- вложенности внутри схемы -->
```

```
<xsl:template name="newline">
  <br/>
  <xsl:for-each select="ancestor::xsd:*[not(self::xsd:schema)]">
    <xsl:text>    </xsl:text>
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Обсуждение

В главе 10 шла речь о репозитории сообщений, где можно сохранить метаданные о сообщениях, которыми обмениваются клиент и сервер. Язык WSDL дает пример синтаксиса для описания такого репозитория в применении к архитектуре на основе Web-сервисов. Один и тот же WSDL-файл можно использовать для генерации кода и документации. В данном рецепте мы занимались только выдачей документации по запросу.

На языке WSDL Web-сервисы описываются в терминах ассоциаций между сервисами и портами. В терминологии WSDL *порт* – это концевая точка, определенная как комбинация привязки и сетевого адреса. *Привязкой* называется конкретный протокол и спецификация формата данных для конкретного типа порта. *Тип порта* определяет абстрактный набор операций, поддерживаемых одной или несколькими концевыми точками. *Операция* – это действие, выполняемое сервисом; определяется она в терминах сообщений с той или иной полезной нагрузкой. Полезная нагрузка описывается XSD-схемой. Ниже приведена часть простого WSDL-файла, в котором описывается набор сервисов, предлагаемых внутри компании Acme Corporation:

```
<definitions name="StockServices" targetNamespace="http://acme.com/
services.wsdl" xmlns:acme="http://acme.com/services.wsdl" xmlns:xsd1=
"http://acme.com/stockquote.xsd" xmlns:soap="http://schemas.xmlsoap.org/
wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">
```

В языке WSDL по умолчанию применяются XSD-схемы для описания типов данных в сообщениях, которыми обмениваются клиент и сервер:

```
  <types>
    <schema targetNamespace="http://acme.com/services.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
     <element name="Ticker">
        <complexType>
          <all>
           <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
     </element>
```

```
<element name="TradePrice">
   <complexType>
     <all>
      <element name="price" type="float"/>
     </all>
   </complexType>
</element>
<element name="CompanyName">
   <complexType>
     <all>
      <element name="company" type="string"/>
     </all>
   </complexType>
</element>
<element name="ServicesInfo">
   <complexType>
     <sequence>
       <element name="Services">
         <complexType>
           <sequence>
                <element name="Service" type="string" minOccurs="0"
                        maxOccurs="unbounded"/>
           </sequence>
         </complexType>
      </element>
      <element name="Ports">
         <complexType>
           <sequence>
             <element name="Port" type="string" minOccurs="0"
                        maxOccurs="unbounded"/>
           </sequence>
         </complexType>
      </element>
      <element name="Bindings">
         <complexType>
           <sequence>
                <element name="Binding" type="string" minOccurs="0"
                        maxOccurs="unbounded"/>
           </sequence>
         </complexType>
      </element>
      <element name="PortTypes">
         <complexType>
           <sequence>
```

```
                    <element name="Port" type="string" minOccurs="0"
                             maxOccurs="unbounded"/>
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
      <element name="ServicesQuery">
        <complexType>
          <all>
              <element name="Service" type="string"/>
              <element name="Port" type="string"/>
              <element name="Binding" type="string"/>
              <element name="PortType" type="string"/>
          </all>
        </complexType>
       </element>
       <element name="ServicesResponse">
        <complexType>
          <sequence>
            <any/>
          </sequence>
        </complexType>
       </element>
    </schema>
  </types>
```

Элементы `message` связывают имена с типами данных и специфицируют, что именно передается в сообщении:

```
<message name="GetLastTradePriceInput">
   <part name="body" element="xsd1:Ticker"/>
</message>
<message name="GetLastTradePriceOutput">
   <part name="body" element="xsd1:TradePrice"/>
</message>
<message name="GetCompanyInput">
   <part name="body" element="xsd1:Ticker"/>
</message>
<message name="GetCompanyOutput">
   <part name="body" element="xsd1:CompanyName"/>
</message>
<message name="GetTickerInput">
   <part name="body" element="xsd1:CompanyName"/>
```

```
</message>
<message name="GetTickerOutput">
   <part name="body" element="xsd1:Ticker"/>
</message>
<message name="GetServicesInput"/>
<message name="GetServicesOutput">
   <part name="body" element="xsd1:ServicesInfo"/>
</message>
<message name="QueryServicesInput">
   <part name="body" element="xsd1:ServicesQuery"/>
</message>
<message name="QueryServicesOutput">
   <part name="body" element="xsd1:ServicesRespose"/>
</message>
```

Тип порта определяет, какие операции доступны в конкретной концевой точке сервиса (порту). В этом рецепте описываются порты двух типов: один предназначен для выдачи информации о курсах акций, а другой описывает службу обнаружения сервисов. Последнюю мы как раз и реализовали.

```
<portType name="StockPortType">
   <operation name="GetLastTradePrice">
      <input message="acme:GetLastTradePriceInput"/>
      <output message="acme:GetLastTradePriceOutput"/>
   </operation>
   <operation name="GetTickerFromCompany">
      <input message="acme:GetTickerInput"/>
      <output message="acme:GetCompanyOutput"/>
   </operation>
   <operation name="GetCompanyFromTicker">
      <input message="acme:GetCompanyInput"/>
      <output message="acme:GetTickerOutput"/>
   </operation>
</portType>
<portType name="ServicePortType">
   <operation name="GetServices">
      <input message="acme:GetServicesInput"/>
      <output message="acme:GetServicesOutput"/>
   </operation>
   <operation name="QueryServices">
      <input message="acme:QueryServicesInput"/>
      <output message="acme:QueryServicesOutput"/>
   </operation>
</portType>
```

Привязки связывают операции с протоколами. В данном случае мы объявляем, что все операции привязаны к протоколу SOAP:

```
<binding name="StockQuoteSoapBinding" type="acme:StockPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
  <operation name="GetLastTradePrice">
     <soap:operation soapAction="http://acme.com/GetLastTradePrice"/>
     <input>
        <soap:body use="literal"/>
     </input>
     <output>
        <soap:body use="literal"/>
     </output>
  </operation>
</binding>
<binding name="StockTickerSoapBinding" type="acme:StockPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
  <operation name="GetTickerFromCompany">
     <soap:operation soapAction="http://acme.com/GetTickerSymbol"/>
     <input>
        <soap:body use="literal"/>
     </input>
     <output>
        <soap:body use="literal"/>
     </output>
  </operation>
</binding>
<binding name="StockNameSoapBinding" type="acme:StockPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
  <operation name="GetCompanyFromTicker">
     <soap:operation soapAction="http://acme.com/GetCompanyName"/>
     <input>
        <soap:body use="literal"/>
     </input>
     <output>
        <soap:body use="literal"/>
     </output>
  </operation>
</binding>
<binding name="ServiceListSoapBinding" type="acme:ServicePortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/
soap/http"/>
```

```
    <operation name="GetServices">
       <soap:operation soapAction="http://acme.com/GetServices"/>
       <input>
          <soap:body use="literal"/>
       </input>
       <output>
          <soap:body use="literal"/>
       </output>
    </operation>
</binding>
```

Ниже объявлено два сервиса. Один относится к курсам акций, другой – к обнаружению сервисов. Обратите внимание, как организован набор портов, описывающих функциональность, доступную с помощью данного сервиса:

```
<service name="StockInfoService">
  <documentation>Предоставляет информацию о курсах акций.</documentation>
  <port name="StockQuotePort" binding="acme:StockQuoteSoapBinding">
    <soap:address location="http://acme.com/stockquote"/>
  </port>
  <port name="StockTickerToNamePort" binding="acme:StockTickerSoapBinding">
    <soap:address location="http://acme.com/tickertoname"/>
  </port>
  <port name="StockNameToTickerPort" binding="acme:StockNameSoapBinding">
      <soap:address location="http://acme.com/nametoticker"/>
  </port>
</service>
<service name="ServiceInfoService">
    <documentation>Предоставляет информацию об имеющихся сервисах.
  </documentation>
  <port name="ServiceListPort" binding="acme:ServiceListSoapBinding">
    <soap:address location="http://acme.com/stockquote"/>
  </port>
  <port name="ServiceQueryPort" binding="acme:ServiceQuerySoapBinding">
    <soap:address location="http://acme.com/tickertoname"/>
  </port>
  </service>

</definitions>
```

Наш CGI-сценарий выводит форму с запросом о сервисах. Вы можете задать параметры запроса, как показано на рис. 13.6.

В ответ на этот запрос будет выдан результат, показанный на рис. 13.7.

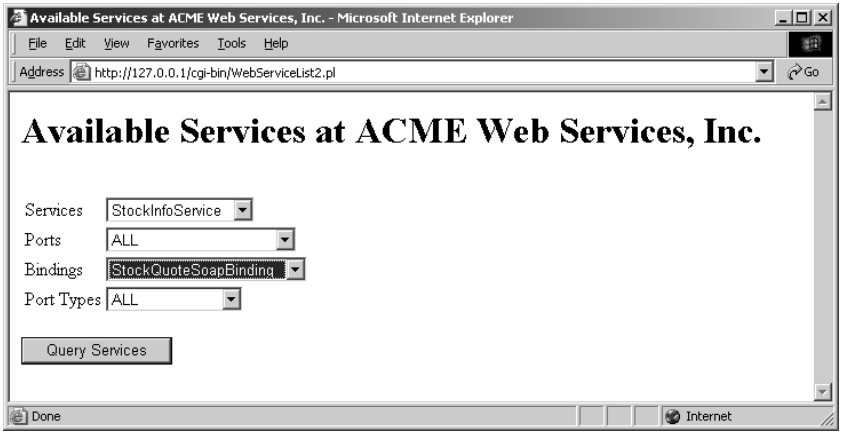


Рис. 13.6. Запрос к WSDL

Рис. 13.7. Ответ на запрос к WSDL

См. также

Спецификация WSDL размещена по адресу *http://www.w3.org/TR/wsdl*. Пока она еще не принята в качестве официальной рекомендации W3C, но язык быстро завоевывает признание и обеспечен поддержкой индустрии. Официальная группа Web Services Description Working Group (*http://www.w3.org/2002/ws/desc/*)

работает над документом, который в конечном итоге будет санкционирован консорциумом W3C.

О том, как выполнять XSLT-преобразования из Perl, не запуская отдельный процесс, см. рецепт 14.6.

Уче Огбуджи (Uche Ogbuji) написал о генерации документации и RDF-файла из WSDL-документа в статье *WSDL processing with XSLT*, которая размещена по адресу *http://www-106.ibm.com/developerworks/library/ws-trans/?dwzone=ws*.

Глава 14. Расширение и встраивание XSLT

Думаю, у каждой женщины должен быть лифчик Wonderbra. Есть столько способов их улучшить... Все так делают.

Кристина Агилера

14.0. Введение

По-настоящему амбициозные программисты никогда не удовлетворены тем, что им дают, и как одержимые стараются улучшить то, что имеют. Я говорю «амбициозные», а не «великие», потому что, на мой взгляд, величие подразумевает понимание того, когда лучше работать в имеющейся системе, а когда ее стоит расширять. Но так или иначе, эта глава посвящена расширению системы как с точки зрения самого языка XSLT, которому бывает нужна функциональность, более удобно реализуемая на других языках, так и с точки зрения других языков, которым бывает необходим XSLT.

Расширение XSLT – это, по определению, механизм на грани спецификации. Расширения делают таблицу стилей менее переносимой. Вы безусловно рискуете, когда используете расширения, предоставляемые конкретным процессором XSLT, или создаете свои собственные. Даже в том случае, когда расширение реализовано на таком переносимом языке, как Java. Наиболее очевидная причина состоит в том, что не все процессоры XSLT написаны на Java и вряд ли когда-нибудь станут поддерживать Java-расширения. Но даже если вы готовы примириться с тем, что ваше расширение будет работать только в процессорах, написанных на Java, опасность все равно существует, так как механизм расширения XSLT в версии 1.0 стандартизован не до конца. В версии 1.1 наметился прогресс в этом отношении, но эта версия больше не является официальной, и потому многие процессоры ее не поддерживают. Как ни странно, в версии XSLT 2.0 был сделан шаг назад по сравнению с 1.1, и метод связывания с функциями расширения оставлен неопределенным.

EXSLT.org – это портал, участники которого стремятся установить стандарты, которым разработчики процессоров XSLT могли бы следовать при реализации общеупотребительных расширений. В главах 2 и 3 проект EXSLT упоминался в связи с расширениями, касающимися математических функций и функций для работы с датами. Но в нем определены и другие категории расширений, и некоторые из них будут кратко рассмотрены в данной главе. На этот сайт обязательно стоит заглянуть, прежде чем приступать к разработке собственного расширения. Велика вероятность, что кто-то его уже написал или думал о том, как оно должно работать.

В противоположность расширению XSLT, встраивание подразумевает выполнение XSLT-преобразований из других языков без запуска процессора XSLT в отдельном процессе. Мы покажем, как это можно сделать из программ, написанных на языках Perl и Java.

Работая над этой главой, я очень скоро осознал, что можно было бы посвятить расширению и встраиванию целую книгу, особенно если сравнивать разные реализации, рассматривать разные языки программирования и приводить содержательные примеры. Чтобы не увеличивать размер этой главы до бесконечности, я решил остановиться только на процессорах Xalan-Java 2 и Saxon и ограничиться преимущественно языками Java и JavaScript. Кроме того, мы обсудим компонент MSXML.

Итак, я расскажу лишь о том, как использовать расширения в процессорах Saxon, Xalan-Java 2 и MSXML.

14.1. Функции расширения в Saxon

XSLT 1.0 (Saxon версия 6.5.4)

Saxon позволяет обращаться к функциям расширения, написанным на Java, следуя интерфейсу, который был определен в проекте стандарта XSLT 1.1.

В настоящее время Saxon в качестве языка расширения поддерживает только Java, поэтому функции расширения объявляются, как в следующем примере:

```
XSLT 1.0 (Saxon version 6.5.4)
<xsl:stylesheet
    version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:Math="java:java.lang.Math"
    exclude-result-prefixes="Math">

    <xsl:script implements-prefix="Math"
                          xmlns:Math="java:java.lang.Math"
                          language="java"
                          src="java:java.lang.Math"/>
```

Принятое здесь соглашение об именовании пространств имен не обязательно. Но, если ему следовать, то элемент `xsl:script` можно опустить. Следовательно, если вы собираетесь вызвать расширение только один раз, то допустимо написать так:

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

    <xsl:variable name="PI" select="4 * Math:atan(1.0)"
                          xmlns:Math="java:java.lang.Math"/>
    <!-- ... -->
    </xsl:stylesheet>
```

Здесь для связывания с реализацией на Java мы употребили пространство имен, а не элемент `xsl:script`. Отметим, что оба способа связывания независимы; если включен элемент `xsl:script`, то пространство имен не имеет значения. Напротив, если элемент `xsl:script` опущен, то за связывание отвечает только пространство имен.

XSLT 2.0 (Saxon версия 8.x)

Механизм вызова Java-функций расширения практически такой же, как в Saxon 6.5.4. Однако элемент `saxon:script` объявлен устаревшим и, согласно информации на сайте, из будущих версий может быть исключен. Предпочтительный способ – объявить на верхнем уровне пространство имен в виде `java:` и далее полностью квалифицированное имя класса.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="java:java.lang.Math"
  exclude-result-prefixes="math">

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

    <xsl:template match="/">
       <pi><xsl:value-of select="4 * math:atan(1.0)"/></pi>
    </xsl:template>

</xsl:stylesheet>
```

14.2. Элементы расширения в Saxon

XSLT 1.0 (Saxon версия 6.5.4)

Элементы расширения в Saxon можно реализовать только на языке Java. Необходимо определить пространство имен, которое связывает имя расширения с его реализацией. Но правила здесь более четкие, чем для функций расширения. URI пространства имен должен завершаться символом `/`, за которым следует полностью квалифицированное имя Java-класса, реализующего интерфейс `com.icl.saxon. ExtensionElementFactory`:

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:acmeX="http://www.acmeX.com/com.acemX.SuperExtensionFactory"
    extension-element-prefixes="acmeX">

<!-- ... -->

</xsl:stylesheet>
```

Префикс должен также входить в список, заданный в значении атрибута `extension-element-prefixes`.

Дополнительные сведения об интерфейсе `ExtensionElementFactory` см. в рецепте 14.15.

XSLT 2.0 (Saxon версия 8.x)

Механизм по существу тот же самый, но имена классов в библиотеке Saxon изменились (например, `net.sf.saxon.style.ExtensionElementFactory`).

14.3. Функции расширения в Xalan-Java 2

XSLT 1.0 (Xalan-Java 2.6.2)

В процессоре Xalan-Java 2 связывание с функциями расширения производится с помощью двух расширений Xalan: `xalan:component` и `xalan:script`, а URI пространства имен, зарезервированного для Xalan, – `http://xml.apache.org/xslt`.

Элемент `xalan:component` ассоциирует префикс пространства имен с именами функций или элементов расширения, которые будут определены в объемлющем элементе `xalan:script`. Этот элемент определяет язык реализации расширения и саму реализацию. Вариантов несколько. Пользователи, которые лишь изредка прибегают к Java-расширениям, должны обратить внимание на то, что Xalan поддерживает сокращенный синтаксис, не требующий элементов `xalan:component` и `xalan:script`. Достаточно объявить пространство имен одним из вышеупомянутых способов и вызвать Java-функцию, применяя подходящий синтаксис. К сценарным языкам этот способ неприменим.

14.4. Описание Java-функций расширения в формате класса

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:Math="xalan://java.lang.Math">

<xalan:component prefix="Math" functions="sin cos tan atan">
    <xalan:script lang="javaclass" src="xalan://java.lang.Math"/>
</xalan:component>

<xsl:variable name="pi" select="4.0 *"/>

<!-- ... -->

</xsl:stylesheet>
```

Если воспользоваться этим способом и опустить элемент `xalan:component`, то таблица стилей может работать и с Saxon, и с Xalan.

14.5. Описание Java-функций расширения в формате пакета

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:myJava="xalan://java.lang">

<xalan:component prefix="Math" functions="sin cos tan atan">
    <xalan:script lang="javaclass" src="java.lang"/>
</xalan:component>

<xsl:variable name="pi" select="4.0 * myJava:Math.atan(1.0)"/>

<!-- ... -->

</xsl:stylesheet>
```

Этот способ удобен, когда нужно сослаться на много классов в одном и том же пакете.

14.6. Описание Java-функций расширения в формате Java

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:java="http://xml.apache.org/xslt/java">

<xalan:component prefix="Math" functions="sin cos tan atan">
    <xalan:script lang="javaclass" src="http://xml.apache.org/xslt/java"/>
</xalan:component>

<xsl:variable name="pi" select="4.0 * java:java.lang.Math:atan(1.0)"/>

<!-- ... -->

</xsl:stylesheet>
```

Этот способ применяется, когда нужно затребовать доступ к большому числу Java-расширений в одном объявлении пространстве имен. Недостаток в том, что для каждого вызова приходится писать больше слов.

14.7. Функции расширения на языке сценариев с использованием встроенного сценария

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:trig="http://www.acmeX.com/extend/trig">

<xalan:component prefix="trig" functions="sin cons tan atan">
    <xalan:script lang="javascript">
            function sin (arg){ return Math.sin(arg);}
            function cos (arg){ return Math.cos(arg);}
            function tan (arg){ return Math.tan(arg);}
            function atan (arg){ return Math.atan(arg);}
    </xalan:script>
</xalan:component>

<xsl:variable name="pi" select="4.0 * trig:atan(1.0)"/>

<!-- ... -->

</xsl:stylesheet>
```

Xalan в настоящее время поддерживает языки JavaScript, NetRexx, BML, JPython, Jacl, JScript, VBScript и PerlScript, но нужные расширения следует брать у сторонних компаний или лиц, обеспечивающих поддержку соответствующего языка. Подробности см. на странице *http://xml.apache.org/xalan-j/extensions.html#supported-lang*.

14.8. Элементы расширения в Xalan-Java 2

Элементы расширения в Xalan можно писать на языке Java или на одном из поддерживаемых сценарных языков. Для Java-расширений допускается сокращенный синтаксис без использования элементов `xalan:component` или `xalan:script`.

14.9. Реализация элемента расширения на языке Java

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:MyExt="xalan://com.AcmeX.MyExtensionElement">
extension-element-prefixes="MyExt">

<xalan:component prefix="MyExt" elements="superExtension">
```

```
    <xalan:script lang="javasclass"
                  src=" xalan:// com.AcmeX.MyExtensionElement"/>
</xalan:component>

<xsl:template match="*">
    <myExt:superExtension attr1="val1" attr2="val2">
           <!-- ... -->
    <myExt:superExtension>
</xsl:template>

</xsl:stylesheet>
```

Реализация должна иметь вид Java-класса, в котором есть метод с такой сигнатурой:

```
public class com.AcmeX.MyExtensionElement
{
   public SomeType superExtension(
       org.apache.xalan.extensions.XSLProcessorContext ctx,
       org.apache.xalan.templates.ElemExtensionCall extensionElement)
    {
           //...
    }
}
```

где `SomeType` – тип возвращаемого значения, `ctx` – объект, представляющий контекст обработки, а `extensionElement` – узел, соответствующий элементу расширения в таблице стилей. В сигнатуре метода можно указывать и суперклассы упомянутых типов. Базовым классом для класса `com.AcmeX.MyExtensionElement` может быть все, что угодно; его может и вовсе не быть, как показано в примере выше.

Результат, который возвращает этот метод, помещается в результирующее дерево; если вы не хотите его модифицировать, задавайте для метода тип `void`. Дополнительную информацию о классах `XSLProcessorContext` и `ElemExtensionCall` см. в рецепте 14.15.

14.10. Реализация элемента расширения на сценарном языке

Расширения на сценарных языках очень напоминают Java-расширения с тем отличием, что расширение реализуется внутри элемента `xalan:script`:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:MyExt="xalan://com.AcmeX.MyExtensionElement">
extension-element-prefixes="MyExt">
```

```
<xalan:component prefix="rep" elements="repeat">
  <xalan:script lang="javascript">
    function superExtension(ctx, elem)
    {
      /* ... */
      return null ;
    }
  </xalan:script>
</xalan:component>

<xsl:template match="*">
  <myExt:superExtension attr1="val1" attr2="val2">
    <!-- ... -->
  <myExt:superExtension>
</xsl:template>
</xsl:stylesheet>
```

Как и в случае Java, возвращенное значение помещается в результирующее дерево, но можно подавить это действие, вернув `null`. Пример см. в рецепте 14.13.

XSLT 2.0

В настоящее время мне неизвестно о каких-либо действиях по модернизации процессора Xalan до уровня XSLT 2.0.

14.11. Функции расширения в MSXML

XSLT 1.0

Компоненты MSXML 3.0, 4.0, равно как и процессор XSLT в .NET, расширяемы на языках Jscript и VBScript. В .NET расширения можно писать и на языке C#. В MSXML для объявления расширений служит элемент `ms:script`:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ms="urn:schemas-microsoft-com:xslt"
    xmlns:myExt="urn:AcmeX.com:xslt">

    <ms:script language="JScript" implements-prefix="myExt">
        <![CDATA[
        function superExtension(ops) {
            /* ... */
            return result;
        }
        ]]>
    </ms:script>

</xsl:stylesheet>
```

XSLT 2.0

В настоящее время корпорация Microsoft не планирует реализовывать XSLT 2.0 на платформе .NET, решив поддерживать только XQuery. Однако ведется активная работа по переносу Saxon на платформу .NET (на языке C#) (*http://sourceforge.net/projects/saxondotnet/* и *http://weblog.saxondotnet.org/*).

См. также

Написанная на C для проекта Gnome библиотека libxslt также поддерживает расширения. См. *http://xmlsoft.org/XSLT/extensions.html*.

14.12. Использование встроенных расширений Saxon и Xalan

Задача

Вы хотите воспользоваться какими-то полезными расширениями, реализованными в этих популярных процессорах XSLT.

Решение

XSLT 1.0

Этот рецепт разбит на ряд мини-рецептов, иллюстрирующих применение наиболее интересных расширений в Saxon и Xalan. Во всех примерах префикс `saxon` ассоциирован с пространством имен http://icl.com/saxon, а префикс `xalan` – с http://xml.apache.org/xslt.

Требуется организовать вывод в несколько файлов

Мы уже несколько раз встречались в этой книге с реализованным в Saxon механизмом вывода результатов в несколько файлов. Для этого предназначен элемент `saxon:output`. Имеется также элемент `xsl:document`, но он будет работать лишь, если атрибут `version` в таблице стилей равен «1.1», а потому его следует избегать. Выходной файл определяется атрибутом `href`, который может быть вычисляемым выражением:

```
<saxon:output href="toc.html">
    <html>
        <head><title>Оглавление</title></head>
        <body>
            <xsl:apply-templates mode="toc" select="*"/>
        </body>
    </html>
</saxon:output>
```

В Xalan принят совершенно другой подход к выводу нескольких файлов. Вместо одной команды Xalan предоставляет целых три: `redirect:open`, `redirect:close`

и redirect:write. С этими элементами расширения ассоциировано пространство имен xmlns:redirect = "org.apache.xalan.xslt.extensions.Redirect". В типичных случаях можно обойтись одним элементом redirect:write, поскольку в отсутствие других команд он откроет файл, запишет в него данные и закроет.

У каждого из упомянутых элементов есть атрибут file и/или select для указания выходного файла. Значением атрибута file является строка, так что с его помощью имя файла можно задать непосредственно. Атрибут select в качестве значения принимает выражение XPath и, стало быть, позволяет формировать имя файла динамически. Если включить оба атрибута, то расширение redirect сначала вычисляет значение select, а если при этом не получается допустимое файла, обращается к атрибуту file.

```
<xalan:write file="toc.html">
    <html>
        <head><title>Оглавление</title></head>
        <body>
            <xsl:apply-templates mode="toc" select="*"/>
        </body>
    </html>
</xalan:write>
```

Пользуясь расширенными возможностями Xalan, вы можете переключиться с основного файла на дополнительные, оставив основной файл открытым. Это, правда, идет вразрез с заложенным в XSLT принципом работы без побочных эффектов, но вероятно, Xalan обеспечивает предсказуемость результата:

```
<xsl:template match="doc">
<xalan:open file="regular.xml"/>
    <xsl:apply-templates select="*"/>
<xalan:close file="regular.xml"/>
<xsl:template/>


<xsl:template match="regular">
    <xalan:write file="regular.xml">
            <xsl:copy-of select="."/>
    </xalan:write/>
</xsl:template>

<xsl:template match="*">
    <xsl:variable name="file" select="concat(local-name( ),'.xml')"/>
    <xalan:write select="$file">
            <xsl:copy-of select="."/>
    </xalan:write/>
</xsl:template>
```

В XSLT 2.0 поддержка вывода в несколько файлов уже встроена в виде нового элемента `xsl:result-document`:

```
<xsl:result-document format="html" href="toc.html">
    <html>
        <head><title>Оглавление</title></head>
        <body>
            <xsl:apply-templates mode="toc" select="*"/>
        </body>
    </html>
</xsl:result-document>
```

Вы хотите расщепить сложное преобразование на несколько более простых, выполняемых последовательно

Разработчики, привыкшие работать в системе Unix, хорошо знакомы с конвейерной обработкой, когда выход одной команды подается на вход другой. Этот механизм имеется и в других операционных системах, в том числе в Windows. Гениальность этой идеи состоит в том, что она позволяет решить сложную задачу, разбив ее на более простые.

Поскольку XSLT-преобразование – это в конечном итоге преобразование одного дерева в другое, то применение конвейера выглядит вполне естественно. Здесь результат одного преобразования становится исходным деревом для следующего. Мы уже не раз встречались с подобными примерами, когда функция расширения `node-set()` создавала промежуточные наборы узлов, обрабатываемые на более поздних стадиях. Saxon предоставляет альтернативный способ с помощью атрибута `saxon:next-in-chain` элемента `xsl:output`. Этот атрибут перенаправляет выход в другую таблицу стилей. Его значением является URL таблицы, которая должна обрабатывать выходной поток. Этот поток должен содержать корректно сформированный XML, а атрибуты, управляющие форматом вывода (например, `method`, `cdata-section-elements`) игнорируются. Вторая таблица стилей выводит данные в то место, куда выводила бы первая, если бы не было атрибута `saxon:next-in-chain`.

В Xalan к аналогичной функциональности принят иной подход; применяется элемент расширения `pipeDocument`, обладающий интересной особенностью – его можно включить в не содержащую больше ничего таблицу стилей и создать тем самым конвейер между независимыми таблицами, ничего не знающими о существовании друг друга. Таким образом реализация Xalan гораздо ближе к конвейеру Unix, так как информация о конвейере не «зашивается» в участвующие в нем таблицы стилей. Предположим, что таблица *strip.xslt* удаляет из XML-документа, описывающего книгу, какие-то элементы, а таблица *contents.xslt* создает оглавление на основе иерархической структуры. Тогда их можно следующим образом объединить в конвейер:

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

```
      xmlns:pipe="xalan://PipeDocument"
      extension-element-prefixes="pipe">

    <xsl:param name="source"/>
    <xsl:param name="target"/>
    <!--Список сохраняемых элементов. Все остальные удаляются. -->
    <xsl:param name="preserve-elems"/>

    <pipe:pipeDocument source="{$source}" target="{$target}">

      <stylesheet href="strip.xslt">
            <param name="preserve-elems" value="{$preserve-elems}"/>
      </stylesheet>

      <stylesheet href="contents.xslt"/>

    </pipe:pipeDocument>

  </xsl:stylesheet>
```

Этот код создаст оглавление только из оставшихся элементов, не жертвуя независимостью таблиц *strip.xsl* и *contents.xsl*.

Требуется работать с датами и временем

В главе 4 приведено много рецептов для работы с датами и временем, но в XSLT нет способа получить текущую дату и время. И в Saxon, и в Xalan реализованы определенные в проекте EXSLT функции из модуля, относящегося к дате и времени. В этом разделе мы для справки приведем соответствующую документацию. Все функции перечислены в таблице 14.1 с указанием возвращаемого значения и аргументов. Вопросительным знаком обозначаются необязательные аргументы.

Таблица 14.1. *Функции для работы с датами и временем, определенные в EXSLT*

Функция	Поведение
`string date: date-time( )`	Возвращает текущие дату и время в виде строки в формате, который XML-схема определяет как лексическое представление типа `xs:dateTime`.
`string date: date(string?)`	Возвращает дату из строки дата/время, переданной в качестве аргумента. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией date:date-time.

Таблица 14.1. *Функции для работы с датами и временем, определенные в EXSLT (продолжение)*

Функция	Поведение
`string date: time(string?)`	Возвращает время из строки дата/время, переданной в качестве аргумента. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: year(string?)`	Возвращает год переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`boolean date: leap-year(string?)`	Возвращает `true`, если год переданной в качестве аргумента даты является високосным. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: month-in-year(string?)`	Возвращает номер месяца из переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`string date: month-name(string?)`	Возвращает полное название месяца из переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`string date: month-abbreviation(string?)`	Возвращает сокращенное название месяца из переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.

Таблица 14.1. *Функции для работы с датами и временем, определенные в EXSLT (продолжение)*

Функция	Поведение
`number date: week-in-year(string?)`	Возвращает номер недели в году для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`. Нумерация недель следует стандарту ISO 8601: первой неделей года считается неделя, содержащая первый четверг, а первым днем недели считается понедельник.
`number date: day-in-year(string?)`	Возвращает номер дня в году для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: day-in-month(string?)`	Возвращает номер дня в месяце из переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: day-of-week-in-month(string?)`	Возвращает номер дня недели в месяце для переданной в качестве аргумента даты (например, 3 для третьего четверга в мае). Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: day-in-week(string?)`	Возвращает номер дня недели для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`string date: day-name(string?)`	Возвращает полное название дня недели для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.

Таблица 14.1. *Функции для работы с датами и временем, определенные в EXSLT (окончание)*

Функция	Поведение
`string date: day-abbreviation(string?)`	Возвращает сокращенное название дня недели для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: hour-in-day(string?)`	Возвращает номер часа суток для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: minute-in-hour(string?)`	Возвращает номер минуты часа для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.
`number date: second-in-minute(string?)`	Возвращает номер секунды для переданной в качестве аргумента даты. Если аргумент опущен, то по умолчанию принимается текущее локальное время, возвращенное функцией `date:date-time`.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:date="http://exslt.org/dates-and-times">

<xsl:output method="html" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:template match="/">
  <html>
    <head><title>Моя безыскусная домашняя страница</title></head>
    <body>
      <h1>Моя безыскусная домашняя страница</h1>
      <div>Сейчас <xsl:value-of select="date:time( )"/> <xsl:value-of
      select="date:date(  )"/>, и эта страница такая же скучная, как
      вчера.</div>
    </body>
  </html>

</xsl:template>

</xsl:stylesheet>
```

В XSLT 2.0 поддержка даты и время уже встроена (см. главу 4), так что эти расширения не нужны.

Требуется более эффективная реализация операций над множествами

В главе 9 рассматривались различные способы реализации теоретико-множественных операций, помимо объединения, которое XPath и так поддерживает с помощью оператора `|`. Эти решения и не самые эффективные, и не слишком наглядные.

И Saxon, и Xalan решают проблему путем реализации операций над множествами, определенных в модуле set проекта EXSLT (см. таблицу 14.2).

Таблица 14.2. *Операции над множествами, определенные в модуле set проекта EXSLT*

Функция	Поведение
`Node-set set: difference(node-set, node-set)`	Возвращает разность двух наборов узлов, то есть множество узлов, присутствующих в первом наборе и отсутствующих во втором.
`Node-set set: intersection(node-set, node-set)`	Возвращает пересечение двух наборов узлов, то есть множество узлов, присутствующих и в первом, и во втором наборе.
`Node-set set: distinct(node-set)`	Возвращает подмножество узлов из переданного набора NS, а именно: узел N отбирается, если в наборе NS нет другого узла с таким же строковым значением, как у N, и предшествующего N в порядке документа.
`Boolean set: has-same-node(node-set, node-set)`	Возвращает `true`, если в первом наборе узлов есть хотя бы один узел, присутствующий во втором наборе. Если наборы не имеют общих узлов, возвращает `false`.
`Node-set set: leading(node-set, node-set)`	Возвращает те узлы из первого набора, которые предшествуют в порядке документа первому узлу второго набора. Если первый узел второго набора не входит в первый набор, возвращается пустой набор узлов. Если второй набор пуст, возвращается первый набор узлов.

Таблица 14.2. *Операции над множествами, определенные в модуле set проекта EXSLT (окончание)*

Функция	Поведение
`Node-set set: trailing(node-set, node-set)`	Возвращает те узлы из первого набора, которые следуют в порядке документа за первым узлом второго набора. Если первый узел второго набора не входит в первый набор, возвращается пустой набор узлов. Если второй набор пуст, возвращается первый набор узлов.

Функция `set:distinct` дает удобный способ удаления дубликатов в предположении, что равенство узлов определено как равенство их строковых значений:

```
<xsl:varible name="firstNames" select="set:destinct(person/firstname)"/>
```

Функции `set:leading` и `set:trailing` могут извлекать узлы, заключенные между двумя заданными. Например, в рецепте 12.9 использовалось сложное выражение для поиска узлов `xslx:elsif` и `xslx:else`, ассоциированных с узлом `xslx:if`. Эти расширения позволяют решить задачу проще:

```
<xsl:apply-templates
     select="set:leading(following-sibling::xslx:else |
     following-sibling::xslx:elsif, following-sibling::xslx:if)"/>
```

Здесь говорится, что надо отобрать всех братьев типа `xslx:else` и `xslx:elseif`, которые следуют за текущим узлом, но предшествуют следующему узлу `xslx:if`.

Требуется получить расширенную информацию об узле во входном дереве

В Xalan есть функции, которые позволяют получить информацию о положении узлов во входном дереве. Saxon 6.5.2 предоставляет только функции `saxon:systemId` и `saxon:lineNumber`. Применяются они, в частности, для отладки. Чтобы ими воспользоваться, нужно установить в `true` атрибут `source_location` элемента `TransformerFactory`. Сделать это позволяет флаг `-L` в командной строке или метод `TransformerFactory.setAttribute( )`.

```
systemId( )
systemId(node-set)
```

Возвращают соответственно системный идентификатор текущего узла и первого узла в наборе.

```
lineNumber( )
lineNumber(node-set)
```

Возвращают соответственно номер строки исходного документа, соответствующий текущему узлу и первому узлу в наборе. Если номер строки неизвестен (например, когда на вход подан документ DOM), возвращают -1.

```
columnNumber( )
columnNumber(node-set)
```

Возвращают соответственно номер колонки исходного документа, соответствующий текущему узлу и первому узлу в наборе. Если номер колонки неизвестен (например, когда на вход подан документ DOM), возвращают -1.

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xalan="http://xml.apache.org/xslt"
 xmlns:info="xalan://org.apache.xalan.lib.NodeInfo">

   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="foo">
      <xsl:comment>foo найдено в строке <xsl:value-of
         select="info:lineNumber( )"/> и колонке <xsl:value-of
         select="info:columnNumber( )"/>.</xsl:comment>
      <!-- ... -->
   </xsl:template>

</xsl:stylesheet>
```

Требуется обратиться к реляционной базе данных

Интерфейс XSLT с реляционными базами данных открывает широчайший спектр новых возможностей. И Saxon, и Xalan поддерживают SQL с помощью расширений. Если вы пишете таблицы стилей, которые модифицируют базу данных, то нарушаете принцип работы без побочных эффектов.

Вот что говорит Майкл Кэй по поводу расширений SQL в Saxon: «Не предполагается, что это программное обеспечение промышленного качества (в проекте заложено много ограничений). Скорее, это иллюстрация того, как можно использовать элементы расширения для обогащения возможностей процессора».

В Saxon для доступа к базе данных есть пять элементов расширения: `sql:connect`, `sql:query`, `sql:insert`, `sql:column` и `sql:close`. Они понятны любому, кто работал с реляционными базами через ODBC или JDBC.

```
<sql:connect driver="jdbc-driver" database="db name" user="user name"
password="user password"/>
```

Открывает соединение с базой данных. Любой атрибут может быть вычисляемым выражением. Атрибут `driver` – это имя класса драйвера JDBC, а `database`

должно быть именем, которое JDBC может ассоциировать с реальной базой данных.

```
<sql:query table="the table" column="column names" where="where clause"
row-tag="row element name" column-tag="column element name" disable-output-
escaping="yes or no"/>
```

Выполняет запрос и записывает результаты в выходное дерево, используя элементы для представления строк и колонок. Имена этих элементов задаются с помощью атрибутов `row-tag` и `col-tag` соответственно. Атрибут `column` может содержать список колонок, а если он равен *, выбираются все колонки.

```
<sql:insert table="table name">
```

Выполняет предложение SQL INSERT. Дочерние элементы (`sql:column`) задают данные, которые нужно добавить в таблицу.

```
<sql:column name="col name" select="xpath expr"/>
```

Должен быть дочерним элементом `sql:insert`. Значение можно задать в атрибуте `select` или путем вычисления дочерних элементов `sql:column`. В обоих случаях разрешается использовать только строковые значения. Никакого способа работы со стандартными типами данных SQL не предусмотрено.

Xalan обеспечивает более развитую поддержку SQL, нежели Saxon. В этой главе мы рассмотрим только основы. В разделе «См. также» приведены ссылки на более подробную информацию. В отличие от Saxon, в Xalan для доступа к базе данных применяются функции расширения.

```
sql:new(driver, db, user, password)
```

Устанавливает соединение.

```
sql:new(nodelist)
```

Устанавливает соединение, используя информацию, находящуюся во входном XML-документе или в таблице стилей. Например:

```
<DBINFO>
    <dbdriver>org.enhydra.instantdb.jdbc.idbDriver</dbdriver>
    <dburl>jdbc:idb:../../instantdb/sample.prp</dburl>
    <user>jbloe</user>
    <password>geron07moe</password>
</DBINFO>

<xsl:param name="cinfo" select="//DBINFO"/>
<xsl:variable name="db" select="sql:new($cinfo)"/>

sql:query(xconObj, sql-query)
```

Отправляет запрос базе данных. Объект `xconObj` – это результат функции `new()`. Функция возвращает *потоковый* результирующий набор в виде узла `row-set`.

Можно перебирать набор строк по одной. При каждом обращении возвращается элемент `row`, так что можно начинать преобразовывать набор строк еще до того, как он получен полностью.

```
sql:pquery(xconObj,sql-query-with-params)
sql:addParameter(xconObj, paramValue)
sql:addParameterFromElement(xconObj,element)
sql:addParameterFromElement(xconObj,node-list)clearParameters(xconObj)
```

Используются совместно для реализации параметризованных запросов. Параметры представляются символами ? в тексте запроса. Различные варианты функции `addParameter( )` подставляют вместо них фактические значения еще до начала выполнения запроса. Чтобы стереть старые значения параметров, пользуйтесь функцией `clearParameters( )`.

```
sql:close(xconObj)
```

Закрывает соединение с базой данных.

Функции `sql:query( )` и `sql:pquery()` возвращают узел типа `Document`, который содержит массив элементов `column-header`, один элемент `row`, который используется повторно, и массив элементов `col`. Каждый элемент `column-header` (по одному на колонку в наборе строк) содержит атрибут `ColumnAttribute` для каждого дескриптора колонки в объекте `ResultSetMetaData`. Каждый элемент `col` содержит текстовый узел с представлением значения этой колонки в текущей строке.

Дополнительную информацию об использовании XSLT для доступа к реляционным базам данных см. в книге Doug Tidwell *XSLT* (O'Reilly, 2001).

Требуется динамически вычислить выражение XPath, созданное на этапе выполнения

И в Saxon, и в Xalan есть очень мощная функция расширения `evaluate`, которая принимает на входе строку и вычисляет ее как выражение XPath. В проекте EXSLT.org также определена более переносимая функция `dyn:evaluate()`. Такое средство рассматривалось при работе над XSLT 2.0, но в тот момент рабочая группа решила не включать ее в стандарт под предлогом, что динамическое вычисление «... оказывает значительное влияние на архитектуру процессора на этапе выполнения, а также на его способность производить статическую оптимизацию».

Динамические возможности удобны при создании таблично управляемой таблицы стилей. Следующая таблица стилей может представить информацию о людях в табличном виде, но ее можно настроить на почти произвольный формат входного XML-документа путем простого изменения данных в таблице:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:saxon="http://icl.com/saxon"
  xmlns:paths="http://www.ora.com/XSLTCookbook/NS/paths"
  exclude-result-prefixes="paths">

<xsl:output method="html"/>
```

```
<!-- Этот параметр служит для задания документа, содержащего таблицу, -->
<!-- которая описывает, как искать информацию о людях -->
<xsl:param name="pathsDoc"/>

<xsl:template match="/">
<html>
  <head>
    <title>Люди</title>
  </head>
  <body>
  <!-- Мы загружаем выражение XPath из таблицы [Symbol_Wingdings_224] -->
  <!-- во внешнем документе -->
  <xsl:variable name="peoplePath"
       select="document($pathsDoc)/*/paths:path[@type='people']/@xpath"/>
    <table>
    <tbody>
      <tr>
        <th>Имя</th>
        <th>Фамилия</th>
      </tr>
      <!-- Динамически вычисляем выражение XPath, которое ищет -->
      <!-- информацию о людях -->
      <xsl:for-each select="saxon:evaluate($peoplePath)">
        <xsl:call-template name="process-person"/>
      </xsl:for-each>
    </tbody>
  </table>
  </body>
</html>
</xsl:template>

<xsl:template name="process-person">
  <xsl:variable name="firstnamePath"
       select="document($pathsDoc)/*/paths:path[@type='first']/@xpath"/>
  <xsl:variable name="lastnamePath"
       select="document($pathsDoc)/*/paths:path[@type='last']/@xpath"/>
  <tr>
    <!-- Динамически вычисляем выражение XPath, которое ищет -->
    <!-- ту информацию о человеке, которую мы хотим обрабатывать -->
    <td><xsl:value-of select="saxon:evaluate($firstnamePath)"/></td>
    <td><xsl:value-of select="saxon:evaluate($lastnamePath)"/></td>
  </tr>
</xsl:template>

</xsl:stylesheet>
```

Если данные о человеке представлены в виде элементов, то можно использовать такую таблицу:

```
<paths:paths xmlns:paths="http://www.ora.com/XSLTCookbook/NS/paths">
    <paths:path type="people" xpath="people/person"/>
    <paths:path type="first" xpath="first"/>
    <paths:path type="last" xpath="last"/>
</paths:paths>
```

А если в виде атрибутов, то такую:

```
<paths:paths xmlns:paths="http://www.ora.com/XSLTCookbook/NS/paths">
    <paths:path type="people" xpath="people/person"/>
    <paths:path type="first" xpath="@first"/>
    <paths:path type="last" xpath="@last"/>
</paths:paths>
```

Требуется изменить значение переменной

Возьмите чуть ли не любую книгу по XSLT, и вы прочтете, что невозможность изменить один раз присвоенное переменной или параметру значение – особенность XSLT, а не дефект. Это действительно так, поскольку позволяет предотвратить целый класс ошибок, делает таблицы стилей более понятными и открывает возможность для некоторых оптимизаций. Но иногда это очень неудобно. Saxon позволяет обойти это препятствие с помощью элемента расширения `saxon:assign`. Применять его можно только к переменным, которые описаны как изменяемые с помощью атрибута расширения `saxon:assignable="yes"`:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:saxon="http://icl.com/saxon"
extension-element-prefixes="saxon">

<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:variable name="countFoo" select="0" saxon:assignable="yes"/>

<xsl:template name="foo">
    <saxon:assign name="countFoo" select="$countFoo + 1"/>
    <xsl:comment>Это <xsl:value-of select="$countFoo"/>-ый вызов шаблона
foo.</xsl:comment>
</xsl:template>

<!-- ... -->

</xsl:stylesheet>
```

Требуется написать полноценную функцию в XSLT 1.0

Многие примеры в этой книге реализованы с помощью именованных шаблонов, которые вызываются командой `xsl:call-template`. Часто такая реализация

неудобна и некрасива, поскольку на самом деле хотелось бы обратиться к коду как к настоящей функции, вызываемой так же легко, как функции, встроенные в XPath. Эта возможность поддерживается в XSLT 2.0, а в 1.0 имеет смысл обратить внимание на EXSLT-расширение `func:function`. которое реализовано в Saxon и в последней версии Xalan (версия 2.3.2 на момент написания книги). Следующий код представляет собой шаблон из главы 2, реализованный в виде функции:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:func="http://exslt.org/functions"
   xmlns:str="http://www.ora.com/XSLTCookbook/namespaces/strings"
   extension-element-prefixes="func">

   <xsl:template match="/">
      <xsl:value-of
         select="str:substring-before-last('123456789a123456789a123',
      </xsl:template>

   <func:function name="str:substring-before-last">
   <xsl:param name="input"/>
   <xsl:param name="substr"/>

  <func:result>
      <xsl:if test="$substr and contains($input, $substr)">
         <xsl:variable name="temp"
                       select="substring-after($input, $substr)" />
         <xsl:value-of select="substring-before($input, $substr)" />
         <xsl:if test="contains($temp, $substr)">
            <xsl:value-of
                  select="concat($substr,
                              str:substring-before-last($temp, $substr))"/>
         </xsl:if>
       </xsl:if>
     </func:result>
   </func:function>

</xsl:stylesheet>
```

XSLT 2.0

Большинство расширений, имеющихся в Saxon 6, сохранились и в Saxon 8. Но некоторые, например `saxon:function()`, в XSLT 2.0 уже не нужны. Есть ряд функций, предназначенных для расширения возможностей XQuery, поскольку такие возможности уже имеются в XSLT. Так, функции `saxon:index( )` и `saxon:find()` по своему назначению аналогичны элементу `xsl:key` и функции

key( ). Однако в Saxon 8 появился ряд новых расширений, которых не было в предыдущей версии.

Требуется получить выражение XPath для доступа к текущему узлу

Функция saxon:path( ) не имеет аргументов и возвращает строку, содержащую выражение XPath, которое ведет к текущему узлу. Это аналог решения на XSLT, которое приведено в рецепте 15.2 для отладочных целей.

Требуется обрабатывать динамические ошибки

Во многих современных языках, например Java и C++, имеется механизм try-throw-catch для обработки динамических (возникающих на этапе выполнения) ошибок. В коммерческую версию Saxon (Saxon-SA) добавлена псевдофункция saxon:try, предоставляющая похожие, хотя и более ограниченные, возможности. Она принимает в качестве первого аргумента выражение. Если при вычислении этого выражения возникает динамическая ошибка (например, деление на 0, ошибка типизации и т.д.), то возвращается значение второго аргумента:

```
<xsl:template match="/">
   <test>
      <xsl:value-of select="saxon:try(*[0], 'Индекс вне диапазона')"/>
   </test>
</xsl:template>
```

Второй аргумент может быть строкой, содержащей сообщение об ошибке (как в примере выше), или некоторым значением по умолчанию. Майкл Кэй называет saxon:try псевдофункцией, потому что она не следует правилам, принятым для обычных функций в XPath. Точнее, она вычисляет второй аргумент, только если вычисление первого привело к ошибке.

Обсуждение

Использование расширений, реализованных конкретным производителем, – это палка о двух концах. С одной стороны, они позволяют писать на языке XSLT код, более простой или быстрый, чем было бы возможно, оставаясь в рамках стандарта. Иногда с их помощью можно даже сделать нечто, что на стандартном XSLT просто не реализуемо. С другой стороны, есть опасность накрепко связать себе с реализацией, будущее которой неопределенно.

Организация EXSLT.org убеждает разработчиков следовать единым соглашениям о наиболее популярных расширениях, поэтому, если есть выбор, то безусловно следует предпочесть решение EXSLT, а не конкретного поставщика.

Другая тактика состоит в том, чтобы вообще избегать расширений, поставляемых в том или ином продукте, и делать все самому. В этом случае вы контролируете исходный код и можете при необходимости перенести его на несколько процессоров. В рецептах 14.2, 14.3 и 14.4 приведены примеры таких доморощенных расширений.

См. также

В этой книге рассмотрены не все расширения, имеющиеся в процессорах Saxon и Xalan. Дополнительную информацию о расширениях Saxon можно найти на странице *http://saxon.sourceforge.net/saxon6.5.4/extensions.html* или *http://www.saxonica.com/ documentation/documentation.html* (для XSLT 2.0), а о расширениях Xalan – на странице *http://xml.apache.org/xalan-j/extensionslib.html*.

14.13. Расширение XSLT с помощью JavaScript

Задача

Требуется реализовать на языке JavaScript функциональность, отсутствующую в XSLT.

Решение

В следующих примерах мы пользуемся имеющейся в процессоре Xalan-Java 2 возможностью вызывать программы на сценарных языках, например JavaScript. Обычно расширения на JavaScript пишутся для того, чтобы вызвать функцию, которой нет ни в XSLT, ни в XPath. Типичный пример – тригонометрические функции:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:trig="http://www.ora.com/XSLTCookbook/extend/trig">

<xsl:output method="text"/>

<xalan:component prefix="trig" functions="sin">
    <xalan:script lang="javascript">
            function sin (arg){ return Math.sin(arg);}
    </xalan:script>
</xalan:component>

<xsl:template match="/">
    Синус 45 градусов равен <xsl:text/>
    <xsl:value-of select="trig:sin(3.14159265 div 4)"/>
</xsl:template>

</xsl:stylesheet>
```

На JavaScript можно также реализовать функции с побочными эффектами и объекты, наделенные состоянием[1]:

[1] Позор мне за такие предложения! Но, если серьезно, выходя за пределы XSLT, вы уже не обязаны играть по его правилам. Впрочем, и отвечать за последствия придется вам.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:count="http://www.ora.com/XSLTCookbook/extend/counter">

<xsl:output method="text"/>

<xalan:component prefix="count"
            functions="counter nextCount resetCount makeCounter">
    <xalan:script lang="javascript">

    function counter(initValue)
    {
       this.value = initValue ;
    }

    function nextCount(ctr)
    {
       return ctr.value++ ;
    }

    function resetCount(ctr, value)
    {
       ctr.value = value  ;
       return "" ;
    }

    function makeCounter(initValue)
    {
       return new counter(initValue) ;
    }

    </xalan:script>
</xalan:component>

<xsl:template match="/">
    <xsl:variable name="aCounter" select="count:makeCounter(0)"/>
    Count: <xsl:value-of select="count:nextCount($aCounter)"/>
    Count: <xsl:value-of select="count:nextCount($aCounter)"/>
    Count: <xsl:value-of select="count:nextCount($aCounter)"/>
    Count: <xsl:value-of select="count:nextCount($aCounter)"/>
    <xsl:value-of select="count:resetCount($aCounter,0)"/>
    Count: <xsl:value-of select="count:nextCount($aCounter)"/>
</xsl:template>

</xsl:stylesheet>
```

В большинстве реализаций этот код напечатает такую последовательность:

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 0
```

А процессор, которые не ожидает побочных эффектов, теоретически может изменить порядок вычислений и не оправдать ваших надежд.

В примере ниже мы обращаемся к регулярным выражениям в JavaScript:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:regex="http://www.ora.com/XSLTCookbook/extend/regex">

<xsl:output method="text"/>

<xalan:component prefix="regex"
    functions="match leftContext rightContext getParenMatch makeRegExp">
  <xalan:script lang="javascript">

    function Matcher(pattern)
    {
           this.re = new RegExp(pattern) ;
           this.re.compile(pattern) ;
           this.result="" ;
           this.left="" ;
           this.right="" ;
    }

    function match(matcher, input)
    {
       matcher.result = matcher.re.exec(input) ;
       matcher.left = RegExp.leftContext ;
       matcher.right = RegExp.rightContext ;
       return matcher.result[0] ;
    }

    function leftContext(matcher)
    {
       return matcher.left ;
    }

    function rightContext(matcher)
```

```
    {
       return matcher.right ;
    }

    function getParenMatch(matcher, which)
    {
       return matcher.result[which] ;
    }

    function makeRegExp(pattern)
    {
       return new Matcher(pattern) ;
    }

    </xalan:script>
</xalan:component>

<xsl:template match="/">
    <xsl:variable name="dateParser"
       select="regex:makeRegExp('(\d\d?)[/-](\d\d?)[/-](\d{4}|\d{2})')"/>
    Сопоставлено: <xsl:value-of
           select="regex:match($dateParser,
                  'родился 05/03/1964 в Нью-Йорке.')"/>
    Слева: <xsl:value-of select="regex:leftContext($dateParser)"/>
    Справа: <xsl:value-of select="regex:rightContext($dateParser)"/>
    Месяц: <xsl:value-of select="regex:getParenMatch($dateParser, 1)"/>
    День: <xsl:value-of select="regex:getParenMatch($dateParser,2)"/>
    Год: <xsl:value-of select="regex:getParenMatch($dateParser,3)"/>
</xsl:template>
</xsl:stylesheet>
```

Этот код выводит следующий результат:

```
Сопоставлено: 05/03/1964
Слева: Я родился
Справа:  в Нью-Йорке.
Месяц: 05
День: 03
Год: 1964
```

Xalan также позволяет кодировать на JavaScript элементы расширения. Ниже приведен элемент, который повторяет свое содержимое n раз. Он мог бы быть полезен для дублирования строк или фрагментов структуры, а также в качестве простого способа организации циклов:

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

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:rep="http://www.ora.com/XSLTCookbook/extend/repeat"
extension-element-prefixes="rep">

<xsl:output method="xml"/>

<xalan:component prefix="rep" elements="repeat">
  <xalan:script lang="javascript">
<![CDATA[
  function repeat(ctx, elem)
  {
     // Получить значение атрибута n в виде целого числа
     n = parseInt(elem.getAttribute("n")) ;
     // Получить объект Transformer, необходимый
     // для вычисления узлов
     xformer = ctx.getTransformer( ) ;
     // Вычислить содержимое повторяемого элемента n раз
     for(var ii=0; ii < n; ++ii)
     {
       node = elem.getFirstChild( ) ;
       while(node)
     {
       node.execute(xformer) ;
       node = node.getNextSibling( ) ;
     }
     }
     // Возвращаемое значение вставляется в выходной документ;
     // чтобы предотвратить этот эффект, вернем null
     return null ;
  }
]]>
  </xalan:script>
</xalan:component>

<xsl:template match="/">
  <tests>
     <!-- Дублирование текста -->
     <test1><rep:repeat n="10">a</rep:repeat></test1>
     <!-- Дублирование структуры -->
     <test2>
     <rep:repeat n="10">
       <Malady>
          <FirstPart>Shim's</FirstPart>
```

```
            <SecondPart>Syndrome</SecondPart>
          </Malady>
        </rep:repeat>
        </test2>
        <!-- Повторное выполнение XSLT-кода -->
        <!--(собственно, то же самое делается в test1 и test2)-->
        <test3>
        <rep:repeat n="10">
          <xsl:for-each select="*">
            <xsl:copy/>
          </xsl:for-each>
        </rep:repeat>
      </test3>
    </tests>
</xsl:template>
</xsl:stylesheet>
```

Обсуждение

Создавать расширения на JavaScript (или ином встраиваемом сценарном языке) очень соблазнительно. Конечно, приходится мысленно переключаться с одного языка на другой, зато не нужно переходить в новую среду разработки и вызывать компилятор. Но, пожалуй, самое большое достоинство таких языков, как JavaScript или VBScript, – их простота[1].

Но у использования расширений на сценарных языках есть и минус – документация о том, как употреблять их в контексте XSLT, скудная. Поэтому имеет смысл отметить несколько моментов. Большая часть этой информации имеется в документах о расширении Xalan (*http://xml.apache.org/xalan-j/extensions.html*), но ее легко не заметить, когда вы торопитесь сделать что-то к сроку.

Во-первых, сценарные расширения поддерживает только версия Xalan-Java, но не Xalan-C++.

Во-вторых, не забудьте добавить пути к файлам bsf.jar и js.jar (для JavaScript) в список путей, по которым ищутся классы. Это можно сделать в командной строке Unix:

```
java -cp /xalan/bin/xalan.jar:/xalan/bin/xercesImpl.jar:/xalan/bin/
bsf.jar: /xalan/
bin/js.jar org.apache.xalan.xslt.Process -in input.xml -xsl trans.xslt
```

или в переменной окружения CLASSPATH:

```
export CLASSPATH=/xalan/bin/xalan.jar:/xalan/bin/xercesImpl.jar:/
xalan/bin/bsf.jar:/xalan/bin/js.jar
```

[1] Честно говоря, строки, написанные мной на JavaScript до того, как я приступил к работе над этой книгой, можно сосчитать по пальцам.

На платформе Windows вместо разделителя-запятой употребляется точка с запятой, а вместо `export – set`.

В-третьих, обратите внимание, что файл *js.jar* не входит в дистрибутив Xalan. Его нужно отдельно скачать с сайта Mozilla.org (*http://www.mozilla.org/rhino/*).

Сконфигурировав окружение, вы должны составить таблицу стилей с учетом требований, которые Xalan предъявляет к расширениям на сценарных языках. Технические подробности см. в начале этой главы.

Реализовывать функции расширения гораздо проще, чем элементы, поэтому примеров, приведенных в разделе «Решение» (вкупе с документацией по Xalan), должно быть достаточно. Далее мы будем говорить только об элементах расширения.

Когда вызывается функция, ассоциированная с элементом расширения, ей автоматически передаются два объекта. Первый – это контекст типа `org.apache.xalan.extensions.XSLProcessorContext`. С его помощью можно добраться до еще нескольких полезных объектов, например, до контекстного узла, объекта `Stylesheet` и объекта-преобразователя. В нем же реализован метод `outputToResultTree(Stylesheet stylesheetTree, java.lang.Object obj)`, который выводит данные в результирующее дерево. Тот факт, что эти объекты, написанные на Java, доступны из JavaScript, – заслуга каркаса Bean Scripting Framework (*http://jakarta.apache.org/bsf/*), код которого находится в файле *bsf.jar*.

Второй объект – это экземпляр класса `org.apache.xalan.templates.ElemExtensionCall`. Он представляет сам элемент расширения. Вы можете получить его атрибуты и дочерние элементы, необходимые сценарию для реализации функциональности расширения. Делается это с помощью стандартных функций работы с DOM, например: `getAttribute()`, `getFirstChild()`, `getLastChild()` и т.д.

Существует не так уж много ограничений на то, что можно делать с помощью сценарного элемента. Но, чтобы понять, как добиться желаемого результата, придется покопаться в исходном коде Xalan-Java и в документации. А вообще-то писать расширения на сценарных языках имеет смысл только для простых задач, поскольку работают они значительно медленнее кода, написанного на Java.

См. также

Авторитетным источником информации о расширяемости Xalan служит документ *http://xml.apache.org/xalan-j/extensionslib.html*.

14.14. Реализация функций расширения на языке Java

Задача

Требуется написать собственное расширение на языке Java.

Решение

Во введении к этой главе мы рассматривали механизм связывания таблицы стилей с расширениями на Java, поэтому сейчас сосредоточимся на примерах.

В главе 2 показано, как преобразовывать числа из одной системы счисления в другую. На Java очень легко реализовать конвертор из десятичной системы в шестнадцатеричную:

```
package com.ora.xsltckbk.util;

public class HexConverter
{

  public static String toHex(String intString)
  {
     try
     {
      Integer temp = new Integer(intString) ;
      return new String("0x").concat(Integer.toHexString(temp.intValue( ))) ;
     }
     catch (Exception e)
     {
      return new String("0x0") ;
     }
  }
}
```

Исходя из того, что значение возвращается с префиксом `0x`, можно сделать вывод, что эта функция, скорее всего, будет использоваться в приложении для генерации кода, например так:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:hex="xalan://com.ora.xsltckbk.util.HexConverter"
exclude-result-prefixes="hex xalan">

<xsl:template match="group">
enum <xsl:value-of select="@name"/>
{
  <xsl:apply-templates mode="enum"/>
} ;
</xsl:template>

<xsl:template match="constant" mode="enum">
  <xsl:variable name="rep">
```

```
      <xsl:call-template name="getRep"/>
    </xsl:variable>
    <xsl:value-of select="@name"/> = <xsl:value-of select="$rep"/>
    <xsl:if test="following-sibling::constant">
      <xsl:text>,</xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="constant">
    <xsl:variable name="rep">
      <xsl:call-template name="getRep"/>
    </xsl:variable>
  const int <xsl:value-of select="@name"/> = <xsl:value-of select="$rep"/> ;
  </xsl:template>

  <xsl:template name="getRep">
    <xsl:choose>
      <xsl:when test="@rep = 'hex'">
        <xsl:value-of select="hex:toHex(@value)"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="@value"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  </xsl:stylesheet>
```

В следующем примере показано, как можно конструировать Java-объекты и вызывать их методы. При преобразовании XML в SVG возникают трудности с размещением текста. В SVG невозможно узнать, сколько места будет занимать выведенная строка. К счастью, такой способ предоставляет Java. Вопрос в том, насколько правильно «мнение» Java о размере строки, нарисованной конкретным шрифтом, с точки зрения SVG. Но все равно идея кажется заманчивой и ее стоит проверить:

```
package com.ora.xsltckbk.util ;
import java.awt.* ;
import java.awt.geom.* ;
import java.awt.font.* ;
import java.awt.image.*;

public class SVGFontMetrics
{
  public SVGFontMetrics(String fontName, int size)
```

```
    {
       m_font = new Font(fontName, Font.PLAIN, size) ;
       BufferedImage bi
            = new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB);
       m_graphics2D = bi.createGraphics( ) ;
    }

    public SVGFontMetrics(String fontName, int size, boolean bold,
                          boolean italic)
    {
       m_font = new Font(fontName, style(bold,italic) , size) ;
       BufferedImage bi
          = new BufferedImage(1, 1, BufferedImage.TYPE_INT_ARGB);
       m_graphics2D = bi.createGraphics( ) ;
    }

    public double stringWidth(String str)
    {
       FontRenderContext frc = m_graphics2D.getFontRenderContext( );
       TextLayout layout = new TextLayout(str, m_font, frc);
       Rectangle2D rect = layout.getBounds( ) ;
       return rect.getWidth( ) ;
    }

    public double stringHeight(String str)
    {
       FontRenderContext frc = m_graphics2D.getFontRenderContext( );
       TextLayout layout = new TextLayout(str, m_font, frc);
       Rectangle2D rect = layout.getBounds( ) ;
       return rect.getHeight( ) ;
    }

    static private int style(boolean bold, boolean italic)
    {
       int style = Font.PLAIN ;
       if (bold) { style |= Font.BOLD;}
       if (italic) { style |= Font.ITALIC;}
       return style ;
    }

    private Font m_font = null ;
    private Graphics2D m_graphics2D = null;
}
```

Здесь для получения нужной информации используются классы `Graphics2D` и `TextLayout` из Java 2 (JDK 1.3.1). Мы реализовали два открытых конструктора: один – для нормального шрифта, другой – для полужирного или курсивного. Два открытых метода – `stringWidth()` и `stringHeight()` – возвращают информацию о том, сколько места будет занимать строка, выведенная заданным в конструкторе шрифтом. Для большинства употребительных шрифтов результат получается довольно точным, но гарантий никаких нет, надо экспериментировать.

Для тестирования мы написали такую таблицу стилей:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xalan="http://xml.apache.org/xslt"
xmlns:font="xalan://com.ora.xsltckbk.util.SVGFontMetrics"
exclude-result-prefixes="font xalan">

<xsl:output method="xml"/>

<xsl:template match="/">
  <svg width="100%" height="100%">
    <xsl:apply-templates/>
  </svg>
</xsl:template>

<xsl:template match="text">
  <xsl:variable name="fontMetrics"
    select="font:new(@font, @size, boolean(@weight), boolean(@stytle))"/>
  <xsl:variable name="text" select="."/>
  <xsl:variable name="width" select="font:stringWidth($fontMetrics, $text)"/>
  <xsl:variable name="height" select="font:stringHeight($fontMetrics, $text)"/>
  <xsl:variable name="style">
     <xsl:if test="@style">
       <xsl:value-of select="concat('font-style:',@style)"/>
     </xsl:if>
  </xsl:variable>
  <xsl:variable name="weight">
     <xsl:if test="@weight">
       <xsl:value-of select="concat('font-weight:',@weight)"/>
     </xsl:if>
  </xsl:variable>
  <g style="font-family:{@font};font-size:{@size};{$style};{$weight}">
     <!-- Вызываем SVGFontMetrics, чтобы нарисовать прямоугольник, -->
     <!-- размеры которого чуть больше ожидаемых размеров текста. -->
      <!-- Корректируем y с учетом размера предшествующего текста. -->
      <rect x="10"
            y="{sum(preceding-sibling::text/@size) * 2}pt"
```

```
            width="{$width + 2}"
            height="{$height + 2}"
            style="fill:none;stroke: black;stroke-width:0.5;"/>
        <!-- Выводит текст, центрируя его в прямоугольнике -->
        <text x="11"
            y="{sum(preceding-sibling::text/@size) * 2 + @size div 2 + 2}pt">
        <xsl:value-of select="."/>
        </text>
    </g>

</xsl:template>

</xsl:stylesheet>
```

Как показано на рис. 14.1, этот тест дает вполне приличные результаты для некоторых часто используемых шрифтов.

Рис. 14.1. Создание охватывающих прямоугольников правильного размера с помощью расширения SVGFontMetrics

```
<TextWidthTest>
  <text font="Serif" size="9">M's are BIG; l's are small;</text>
  <text font="Serif" size="10">SVG makes handling text no fun at all</text>
  <text font="Helvetica" size="12">But if I cheat with a little Java</text>
  <text font="Arial" size="14" weight="bold">PROMISE ME YOU WON'T TELL
MY MAMMA!
  </text>
  <text font="Century" size="16" style="italic">But if you do, I won't
lose cheer.
  </text>
```

```
  <text font="Courier New" size="18" weight="bold" style="italic">Its
really my tech editor that I fear!</text>
</TextWidthTest>
```

Обсуждение

Примеры из раздела «Решение» работают без изменения и в Xalan, и в Saxon (несмотря на пространство имен `xml.apache.org/xslt`). Объясняется это тем, что мы воспользовались сокращенным синтаксисом объявления Java-класса в пространстве имен.

Отметим, что доступ к конструкторам производится с помощью функции `new()`, и процессоры XSLT понимают, какой из перегруженных конструкторов вызывать, анализируя аргументы. Методам экземпляра Java-класса передается дополнительный (первый по счету) параметр, соответствующий `this`. На примере `HexConverter` видно, что при вызове статических методов параметр `this` не передается.

Пример `SVGFontMetrics` не работает с более старыми версиями JDK, но аналогичные результаты можно получить, используя класс `java.awt.FontMetrics` в сочетании с классом `java.awt.Graphics`:

```
package com.ora.xsltckbk.util ;
import java.awt.* ;
import java.awt.geom.* ;
import java.lang.System ;

public class FontMetrics
{
    public FontMetrics(String fontName, int size)
    {
       // Подойдет любой конкретный класс
       Label component = new Label( ) ;
       m_metrics
            = component.getFontMetrics(
               new Font(fontName, Font.PLAIN, size)) ;
       m_graphics = component.getGraphics( ) ;
    }

    public FontMetrics(String fontName, int size, boolean bold, boolean italic)
    {
       // Подойдет любой конкретный класс
       Label component = new Label( ) ;
       m_metrics
            = component.getFontMetrics(
               new Font(fontName, style(bold,italic) , size)) ;
       m_graphics = component.getGraphics( ) ;
```

```
    }

    // Простой, но менее точный для некоторых шрифтов метод
    public int stringWidth(String str)
    {
            return  m_metrics.stringWidth(str) ;
    }

    // Для большинства шрифтов оказывается точнее
    public double stringWidthImproved(String str)
    {
       Rectangle2D rect = m_metrics.getStringBounds(str, m_graphics) ;
       return rect.getWidth( ) ;
    }

    static private int style(boolean bold, boolean italic)
    {
       int style = Font.PLAIN ;
       if (bold) { style |= Font.BOLD;}
       if (italic) { style |= Font.ITALIC;}
       return style ;
    }

    private java.awt.FontMetrics m_metrics = null;
    private java.awt.Graphics m_graphics = null ;
}
```

Хотя эти примеры могут и не отвечать вашим потребностям, с их помощью мы продемонстрировали, как можно написать собственную функцию расширения на языке Java. Перечислим другие возможности, в порядке возрастания сложности:

1. Использовать вместо `xsl:key` имеющийся в библиотеке Java класс `Hashtable`. Он обеспечивает более точный контроль над тем, какие элементы индексируются, и позволяет изменять индекс во время выполнения. В определении `xsl:key` нельзя ссылаться на переменные, а этот способ не подвержен такому ограничению. Им можно воспользоваться для построения индекса над несколькими документами.
2. Реализовать функцию сортировки узлов, которая компенсирует ограничения `xsl:sort`, например, позволяет сортировать, применяя правила иностранного языка. Дуг Тидвелл (Doug Tidwell) демонстрирует такую технику для процессора Saxon в книге *XSLT* (O'Reilly, 2001).
3. Читать и выводить документы в разных форматах в одной таблице стилей. Например, таким образом можно, помимо XML, читать текстовые файлы, файлы в формате CSV или в двоичном формате. В этом направлении XSLT 2.0 предоставляет элемент `xsl:result-document`.

4. Извлекать сжатый XML-документ непосредственно из zip-файла с помощью класса `java.util.zip.ZipFile`. Полезно было бы изучить исходный код функции `document` имеющегося у вас процессора XSLT.

См. также

В главе 9 мы отложили решение задачи размещения текста в узлах сгенерированного SVG-дерева. Можно было бы применить для этой цели функцию `SVGFontMetrics`.

Хотя это и не имеет непосредственного отношения к расширениям XSLT, разработчикам, интересующимся Java и SVG, было бы полезно познакомиться с системой Batik (*http://xml.apache.org/batik/index.html*).

14.15. Реализация элементов расширения на языке Java

Задача

Требуется расширить функциональность XSLT за счет добавления нестандартных элементов.

Решение

Выше мы показали, как можно воспользоваться расширениями, уже реализованными в процессоре XSLT. А в этом разделе разработаем с нуля собственные элементы расширения. В отличие от функций расширения, для этого потребуется более близкое знакомство с деталями реализации конкретного процессора. Поскольку внутренняя архитектура у всех процессоров разная, большая часть кода окажется непереносимой.

Мы начнем с простого расширения, которое можно назвать, скорее, синтаксическим украшением, а не дополнительной функциональностью. При написании XSLT-кода часто возникает необходимость переключить контекст на другой узел. Идиоматический способ добиться этой цели дает команда `xsl:for-each`. Выглядит это странно, поскольку нам нужно не организовать цикл, а просто сделать контекстным *единственный* узел, определенный атрибутом `select`:

```
<xsl:for-each select="document('new.xml')">
    <!-- Обработать новый документ -->
</xsl:for-each>
```

Мы реализуем элемент расширения `xslx:set-context`, который будет вести себя точно так же, как `xsl:for-each`, но только для первого узла из набора, определенного с помощью `select` (впрочем, обычно в наборе и бывает всего один узел).

Saxon требует, чтобы все элементы расширения, ассоциированные с конкретным пространством имен, реализовывали интерфейс `com.icl.saxon.style.ExtensionElementFactory`. Фабрика отвечает за создание элементов расширения по локальному имени.

```
package com.ora.xsltckbk;
import com.icl.saxon.style.ExtensionElementFactory;
import org.xml.sax.SAXException;

public class CkBkElementFactory implements ExtensionElementFactory {

    public Class getExtensionClass(String localname)  {
       if (localname.equals("set-context")) return CkBkSetContext.class;
       if (localname.equals("templtext")) return CkBkTemplText.class;
       return null;
    }

}
```

При использовании расширения в таблице стилей нужно указывать пространство имен, в котором вслед за завершающим символом / идет полностью квалифицированное имя фабрики. Кроме того, префикс этого пространства имен должен быть указан в атрибуте `extension-element-prefixes` элемента `xsl:stylesheet`:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xslx="http://com.ora.xsltckbk.CkBkElementFactory"
 extension-element-prefixes="xslx">

<xsl:template match="/">
    <xslx:set-context select="foo/bar">
           <xsl:value-of select="."/>
    </xslx:set-context>
</xsl:template>

</xsl:stylesheet>
```

Класс, реализующий элемент `set-context`, наследует классу `com.icl.saxon.style.StyleElement` и обязан предоставить методы `prepareAttributes()` и `process()`. Обычно реализуются также и другие методы, перечисленные в таблице 14.3.

Таблица 14.3. *Важные методы класса* `StyleElement` *из процессора Saxon*

Метод	Назначение
`isInstruction( )`	Расширения всегда возвращают `true`.
`mayContainTemplateBody( )`	Возвращает `true`, если у этого элемента могут быть дочерние. Часто `true` возвращается потому, что существует дочерний элемент `xsl:fallback`.

Таблица 14.3. *Важные методы класса* `StyleElement` *из процессора Saxon (окончание)*

Метод	Назначение
`prepareAttributes( )`	Вызывается на этапе компиляции, чтобы класс мог разобрать информацию, содержащуюся в атрибутах элемента расширения. В этот момент можно также провести локальный контроль.
`validate( )`	Вызывается на этапе компиляции после того, как для всех элементов таблицы стилей выполнен локальный контроль. Позволяет выполнить глобальный контроль с учетом родителей и потомков элемента.
`process(Context context)`	Вызывается на этапе выполнения, чтобы сделать то, для чего элемент предназначен. Этот метод может читать или изменять данные в своем контексте, но не должен модифицировать дерево таблицы стилей.

Реализовать элемент `xslx:set-context` оказалось легко, мы просто позаимствовали код из входящего в Saxon класса `XSLForEach`, изменив его так, чтобы действие выполнялось только один раз:

```
public class CkBkSetContext extends com.icl.saxon.style.StyleElement {

    Expression select = null;

    public boolean isInstruction( ) {
           return true;
    }

    public boolean mayContainTemplateBody( ) {
           return true;
    }
```

Затем мы проверяем наличие атрибута `@select`. Если он имеется, вызываем метод `makeExpression`, который из его строкового значения создает выражение XPath:

```
    public void prepareAttributes( )
        throws TransformerConfigurationException {

      StandardNames sn = getStandardNames( );
      AttributeCollection atts = getAttributeList( );

      String selectAtt = null;

      for (int a=0; a<atts.getLength( ); a++) {
         int nc = atts.getNameCode(a);
         int f = nc & 0xfffff;
```

```
            if (f == sn.SELECT) {
               selectAtt = atts.getValue(a);
            } else {
               checkUnknownAttribute(nc);
            }
         }

         if (selectAtt=  =null) {
            reportAbsence("select");
         } else {
            select = makeExpression(selectAtt);
         }
      }

      public void validate( ) throws TransformerConfigurationException {
          checkWithinTemplate( );
      }
```

Далее следует практически такой же код, как в элементе `for-each` из Saxon, только вместо того чтобы вызывать метод `selection.hasMoreElements` в цикле, мы проверяем наличие дочерних элементов только один раз, выбираем элемент, устанавливаем контекст и текущий узел, обрабатываем дочерние элементы и возвращаем результат в контекст:

```
      public void process(Context context) throws TransformerException
      {
             NodeEnumeration selection = select.enumerate(context, false);
             if (!(selection instanceof LastPositionFinder)) {
                selection = new LookaheadEnumerator(selection);
             }

             Context c = context.newContext( );
             c.setLastPositionFinder((LastPositionFinder)selection);
             int position = 1;

                 if (selection.hasMoreElements( )) {
                    NodeInfo node = selection.nextElement( );
                    c.setPosition(position++);
                    c.setCurrentNode(node);
                    c.setContextNode(node);
                    processChildren(c);
                    context.setReturnValue(c.getReturnValue( ));
             }
      }
}
```

Следующий пример не так прост, поскольку он расширяет возможности XSLT, а не просто слегка модифицирует уже существующую функциональность.

Вы, конечно, понимаете, что раз уж я посвятил целую главу генерации кода, значит, эта тема меня интересует. Но, если XSLT почти оптимален в плане манипуляции XML-документами, то его средства вывода оставляют желать лучшего из-за многословности XML. Рассмотрим задачу генерации простого C++-кода на стандартном XSLT:

```
<classes>
    <class>
           <name>MyClass1</name>
    </class>

    <class>
           <name>MyClass2</name>
    </class>

    <class>
           <name>MyClass3</name>
           <bases>
                  <base>MyClass1</base>
                  <base>MyClass2</base>
           </bases>
    </class>

</classes>
```

Таблица стилей для преобразования этого документа в код на C++ могла бы выглядеть так:

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

<xsl:output method="text"/>

<xsl:template match="class">
class <xsl:value-of select="name"/> <xsl:apply-templates select="bases"/>
{
public:

    <xsl:value-of select="name"/>( ) ;
    ~<xsl:value-of select="name"/>( ) ;
    <xsl:value-of select="name"/>(const <xsl:value-of select="name"/>
&amp;amp; other) ;
    <xsl:value-of select="name"/>&amp;amp; operator =(const <xsl:value-of
select="name"/>&amp;amp; other) ;
} ;
```

```
</xsl:template>

<xsl:template match="bases">
<xsl:text>: public </xsl:text>
<xsl:for-each select="base">
    <xsl:value-of select="."/>
    <xsl:if test="position( ) != last( )">
           <xsl:text>, public </xsl:text>
    </xsl:if>
</xsl:for-each>
</xsl:template>

<xsl:template match="text( )"/>

</xsl:stylesheet>
```

Такую таблицу утомительно писать и трудно читать, так как код на C++ теряется в дебрях разметки.

Расширение `xslx:templtext` решает эту проблему путем создания альтернативной реализации элемента `xsl:text`, который может содержать специальные *символы экранирования*, указывающие на необходимость особой обработки. Экранирование обозначается обратной косой чертой (`\`) и может применяться в двух формах. Очевидная альтернатива – использовать фигурные скобки `{` и `}`, имитирующие синтаксис вычисляемых выражений и XQuery. Но, поскольку эти символы часто употребляются в генераторах кода для других целей, я остановился на `\`.

Форма экранирования	Эквивалентный XSLT-код
\выражение\ \ выражение%разделитель\[1]	<xsl:value-of select="expression"/> <xsl:for-each select="expression"> <xsl:value-of select="."/> <xsl:if test="position( ) != last( )> <xsl:value-of select="delimit"/> </xsl:if> </xsl:for-each>

Имея такой механизм, генератор кода можно записать следующим образом:

```
<xsl:stylesheet
 version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xslx="http://com.ora.xsltckbk.CkBkElementFactory"
 extension-element-prefixes="xslx">

<xsl:output method="text"/>
```

[1] В XSLT 2.0 аналогичную функциональность предоставляет конструкция ***<xsl:value-of select="expression" separator="delimit" />***.

```
<xsl:template match="class">
<xslx:templtext>
class \name\ <xsl:apply-templates select="bases"/>
{
public:

    \name\( ) ;
    ~\name\( ) ;
    \name\(const \name\&amp;amp; other) ;
    \name\&amp;amp; operator =(const \name\&amp;amp; other) ;
} ;
</xslx:templtext>
</xsl:template>

<xsl:template match="bases">
<xslx:templtext>: public \base%', public '\</xslx:templtext>
</xsl:template>

<xsl:template match="text( )"/>

</xsl:stylesheet>
```

Такой код значительно проще писать и читать. Конструкция применима в любом контексте, где нужно генерировать большой объем стандартного кода. Ревнителям чистоты XSLT это расширение может не понравиться, потому что оно вводит в XSLT чуждый ему синтаксис, не имеющий отношения к манипуляциям с XML. Возражение справедливо, но, если взглянуть с практической точки зрения, то многие разработчики отказались бы от применения XSLT (в пользу Perl) для генерации стандартного кода только потому, что ему недостает краткого и не мешающего восприятию текста синтаксиса. Ладно, хватит ходить вокруг да около, пора уже привести код:

```
package com.ora.xsltckbk;
import java.util.Vector ;
import java.util.Enumeration ;
import com.icl.saxon.tree.AttributeCollection;
import com.icl.saxon.*;
import com.icl.saxon.expr.*;
import javax.xml.transform.*;
import com.icl.saxon.output.*;
import com.icl.saxon.trace.TraceListener;
import com.icl.saxon.om.NodeInfo;
import com.icl.saxon.om.NodeEnumeration;
import com.icl.saxon.style.StyleElement;
import com.icl.saxon.style.StandardNames;
```

```
import com.icl.saxon.tree.AttributeCollection;
import com.icl.saxon.tree.NodeImpl;
```

Сначала объявим константы, используемые в простом конечном автомате, который разбирает текст с символами экранирования:

```
public class CkBkTemplText extends com.icl.saxon.style.StyleElement
{
    private static final int SCANNING_STATE = 0 ;
    private static final int FOUND1_STATE   = 1 ;
    private static final int EXPR_STATE     = 2 ;
    private static final int FOUND2_STATE   = 3 ;
    private static final int DELIMIT_STATE  = 4 ;
...
```

Затем определим четыре закрытых класса, реализующих миниязык, на котором написан текст внутри элемента `xslx:templtext`. Базовый класс `CkBkTemplParam` запоминает текст, предшествующий символу экранирования:

```
  private class CkBkTemplParam
  {
    public CkBkTemplParam(String prefix)
    {
       m_prefix = prefix ;
    }

    public void process(Context context) throws TransformerException
    {
       if (!m_prefix.equals(""))
       {
          Outputter out = context.getOutputter( );
          out.setEscaping(false);
          out.writeContent(m_prefix);
          out.setEscaping(true);
       }
    }

    protected String m_prefix ;
  }
```

Класс `CkBkValueTemplParam` наследует `CkBkTemplParam` и обрабатывает простые экранированные выражения вида `\expr\`. Чтобы упростить реализацию, предположим, что внутри элемента `xslx:templtext` экранирование выходной информации отключено:

```
  private class CkBkValueTemplParam extends CkBkTemplParam
  {
```

```
    public CkBkValueTemplParam(String prefix, Expression value)
    {
       super(prefix) ;
       m_value = value ;
    }

    public void process(Context context) throws TransformerException
    {
       super.process(context) ;
       Outputter out = context.getOutputter( );
       out.setEscaping(false);
       if (m_value != null)
       {
          m_value.outputStringValue(out, context);
       }
       out.setEscaping(true);
    }

    private Expression m_value ;
}
```

Класс `CkBkTemplParam` обрабатывает выражения вида `\expr%delimit\`, его код в основных чертах заимствован из класса `XslForEach` в Saxon:

```
private class CkBkListTemplParam extends CkBkTemplParam
{
   public CkBkListTemplParam(String prefix, Expression list,
         Expression delimit)
   {
      super(prefix) ;
      m_list = list ;
      m_delimit = delimit ;
   }

   public void process(Context context) throws TransformerException
   {
      super.process(context) ;
      if (m_list != null)
      {
         NodeEnumeration m_listEnum = m_list.enumerate(context, false);

         Outputter out = context.getOutputter( );
         out.setEscaping(false);
         while(m_listEnum.hasMoreElements( ))
         {
```

```
            NodeInfo node = m_listEnum.nextElement( );
            if (node != null)
            {
               node.copyStringValue(out);
            }
            if (m_listEnum.hasMoreElements( ) && m_delimit != null)
            {
            m_delimit.outputStringValue(out, context);
            }
         }
         out.setEscaping(true);
      }
    }

    private Expression m_list = null;
    private Expression m_delimit = null ;
}
```

Последний закрытый класс `CkBkStyleTemplParam` используется для хранения элементов, вложенных в `xslx:templtext`, например `xsl:apply-templates`:

```
private class CkBkStyleTemplParam extends CkBkTemplParam
{
  public CkBkStyleTemplParam(StyleElement snode)
  {
    m_snode = snode ;
  }

  public void process(Context context) throws TransformerException
  {
    if (m_snode.validationError != null)
    {
      fallbackProcessing(m_snode, context);
    }
    else
    {
        try
      {
        context.setStaticContext(m_snode.staticContext);
        m_snode.process(context);
      }
      catch (TransformerException err)
      {
        throw snode.styleError(err);
```

```
        }
      }
    }
  }
```

Следующие три метода стандартны. Если вы позволяете использовать атрибут `disable-output-escaping` для управления экранированием выходного текста, то его значение следует запомнить в методе `prepareAttributes()`. Образец исходного кода можно найти в файле *XslText.java*:

```
public boolean isInstruction( )
{
  return true;
}

public boolean mayContainTemplateBody( )
{
  return true;
}

public void prepareAttributes( ) throws TransformerConfigurationException
{
  StandardNames sn = getStandardNames( );
  AttributeCollection atts = getAttributeList( );
  for (int a=0; a<atts.getLength( ); a++)
  {
    int nc = atts.getNameCode(a);
    checkUnknownAttribute(nc);
  }
}
```

Этап контроля – подходящее место для анализа содержимого элемента `xslx:templtext` на предмет наличия символов экранирования. Каждый текстовый узел передается анализатору. Стилистические дочерние элементы преобразуются в объекты типа `CkBkStyleTemplParam`. Член `m_TemplParms` – это вектор, в котором хранятся результаты разбора:

```
public void validate( ) throws TransformerConfigurationException
{
  checkWithinTemplate( );
  m_TemplParms = new Vector( ) ;

  NodeImpl node = (NodeImpl)getFirstChild( );
  String value ;
  while (node!=null)
  {
```

```
      if (node.getNodeType( ) == NodeInfo.TEXT)
      {
         parseTemplText(node.getStringValue( )) ;
      }
      else
      if (node instanceof StyleElement)
      {
            StyleElement snode = (StyleElement) node;
           m_TemplParms.addElement(new CkBkStyleTemplParam(snode)) ;
      }
      node = (NodeImpl)node.getNextSibling( );
   }
}
```

Метод `process` обходит вектор `m_TemplParms` и для каждого элемента вызывает его метод `process`:

```
public void process(Context context) throws TransformerException
{
   Enumeration iter = m_TemplParms.elements( ) ;
   while (iter.hasMoreElements( ))
   {
      CkBkTemplParam param = (CkBkTemplParam) iter.nextElement( ) ;
      param.process(context) ;
   }
}
```

Следующие закрытые методы реализуют простой лексический анализатор на базе конечного автомата, который было бы проще написать при наличии аппарата регулярных выражений (который включен в JDK, начиная с версии 1.4.1). Два подряд идущих символа \\ интерпретируются как один литеральный символ \. Аналогично последовательность %% преобразуется в один символ %:

```
private void parseTemplText(String value)
{
   // Этот конечный автомат разбирает текст, распознавая в нем параметры
   int ii = 0 ;
   int len = value.length( ) ;

   int state = SCANNING_STATE ;
   StringBuffer temp = new StringBuffer("") ;
   StringBuffer expr = new StringBuffer("") ;
   while(ii < len)
   {
      char c = value.charAt(ii++) ;
      switch (state)
```

```
 {
   case SCANNING_STATE:
   {
     if (c == '\\')
     {
       state = FOUND1_STATE ;
     }
     else
     {
       temp.append(c) ;
     }
}
break ;

case FOUND1_STATE:
{
   if (c == '\\')
   {
     temp.append(c) ;
     state = SCANNING_STATE ;
   }
   else
   {
     expr.append(c) ;
     state = EXPR_STATE ;
   }
}
break ;

case EXPR_STATE:
{
   if (c == '\\')
   {
     state = FOUND2_STATE ;
   }
   else
   {
     expr.append(c) ;
   }
}
break ;

case FOUND2_STATE:
{
   if (c =  = '\\')
```

```
        {
          state = EXPR_STATE ;
          expr.append(c) ;
        }
        else
        {
          processParam(temp, expr) ;
          state = SCANNING_STATE ;
          temp = new StringBuffer("") ;
                temp.append(c) ;
          expr = new StringBuffer("") ;
        }
      }
      break ;
    }
      }
    if (state == FOUND1_STATE || state == EXPR_STATE)
    {
      compileError("xslx:templtext dangling \\");
    }
    else
    if (state == FOUND2_STATE)
    {
      processParam(temp, expr) ;
    }
    else
    {
      processParam(temp, new StringBuffer("")) ;
    }
}

private void processParam(StringBuffer prefix, StringBuffer expr)
{
   if (expr.length( ) == 0)
   {
      m_TemplParms.addElement(new CkBkTemplParam(new String(prefix))) ;
   }
   else
   {
      processParamExpr(prefix, expr) ;
   }
}

private void processParamExpr(StringBuffer prefix, StringBuffer expr)
{
```

```
int ii = 0 ;
int len = expr.length( ) ;

int state = SCANNING_STATE ;
StringBuffer list = new StringBuffer("") ;
StringBuffer delimit = new StringBuffer("") ;
while(ii < len)
{
   char c = expr.charAt(ii++) ;
   switch (state)
   {
      case SCANNING_STATE:
      {
         if (c == '%')
         {
            state = FOUND1_STATE ;
         }
         else
         {
            list.append(c) ;
         }
      }
      break ;

      case FOUND1_STATE:
      {
         if (c == '%')
         {
            list.append(c) ;
            state = SCANNING_STATE ;
         }
         else
         {
            delimit.append(c) ;
            state = DELIMIT_STATE ;
         }
      }
      break ;

      case DELIMIT_STATE:
      {
         if (c == '%')
         {
            state = FOUND2_STATE ;
         }
```

```
            else
            {
               delimit.append(c) ;
            }
          }
          break ;
        }
      }
      try
      {
         if (state == FOUND1_STATE)
        {
            compileError("xslx:templtext trailing %");
        }
        else
        if (state == FOUND2_STATE)
        {
            compileError("xslx:templtext extra %");
        }
        else
        if (state == SCANNING_STATE)
        {
            String prefixStr = new String(prefix) ;
            Expression value = makeExpression(new String(list)) ;
            m_TemplParms.addElement(
                  new CkBkValueTemplParam(prefixStr, value)) ;
        }
        else
        {
            String prefixStr = new String(prefix) ;
            Expression listExpr = makeExpression(new String(list)) ;
            Expression delimitExpr = makeExpression(new String(delimit)) ;
            m_TemplParms.addElement(
               new CkBkListTemplParam(prefixStr, listExpr, delimitExpr)) ;
         }
      }
      catch(Exception e)
      {
      }
   }

   // Вектор CbkTemplParms, полученный в результате разбора текста
   private Vector m_TemplParms = null;
}
```

В реализацию элемента `xslx:templtext` можно внести несколько полезных дополнений. Например, распространить экранирование списка на несколько списков (`/expr1%delim1%expr2%delim2/`). Это примерно соответствует такому коду на XSLT:

```
<xsl:for-each select="expr1">
    <xsl:variable name="pos" select="position( )"/>
    <xsl:value-of select="."/>
    <xsl:if test="$pos != last( )">
           <xsl:value-of select="delim1"/>
    </xsl:if>
    <xsl:value-of select="expr2[$pos]"/>
    <xsl:if test="$pos != last( )">
           <xsl:value-of select="delim2"/>
    </xsl:if>
</xsl:for-each>
```

Эта возможность была бы полезна для подстановки в текст пары последовательных списков. Например, рассмотрим параметры написанной на C++ функции. Каждый из них представляет собой пару «имя-тип». XSLT-код дает лишь грубое приближение к этой семантике, поскольку предполагается, что наборы узлов, заданные с помощью `expr1` и `expr2`, содержат одинаковое количество элементов. Я полагаю, что в правильная реализация должна была бы подставлять в списки элементы, пока хотя бы в одном наборе остаются узлы, подавляя вывод разделителей для тех списков, в которых узлов уже не осталось. Еще лучше, если бы выбором поведения можно было управлять с помощью атрибутов элемента `xslx:templtext`.

Обсуждение

Недостаток места не позволяет мне привести полную реализацию этих элементов расширения для процессора Xalan. Но, памятуя о написанном во введении, должно быть понятно, как это сделать.

См. также

Тем, кто интересуется расширением Saxon, было бы полезно прочитать статью Майкла Кэя о внутренней архитектуре Saxon (*http://www-106.ibm.com/developerworks/library/x-xslt2*).

14.16. Работа с XSLT в программах на языке Perl

Задача

Имеется задача, для решения которой больше подходит Perl, но какие-то части проще реализовать с использованием XSLT.

Решение

Существует несколько способов воспользоваться XSLT из Perl. Модули `XML::LibXSLT` и `XML::LibXML` предоставляют интерфейс к библиотекам GNOME, реализующим процессоры SAX и XSLT. В следующем примере, заимствованном из книги Erik T. Ray и Jason McIntosh *Perl and XML* (O'Reilly, 2002), приведена Perl-программа для пакетной обработки нескольких XML-файлов одним и тем же XSLT-сценарием, который компилируется только один раз:

```
use XML::LibXSLT;
use XML::LibXML;

# аргументами являются имена таблицы стилей и входных файлов
my( $style_file, @source_files ) = @ARGV;

# инициализировать анализатор и процессор XSLT
my $parser = XML::LibXML->new( );
my $xslt = XML::LibXSLT->new( );
my $stylesheet = $xslt->parse_stylesheet_file( $style_file );

# для каждого входного файла: разобрать, преобразовать, напечатать результат
foreach my $file ( @source_files ) {
    my $source_doc = $parser->parse_file( $source_file );
    my $result = $stylesheet->transform( $source_doc );
    print $stylesheet->output_string( $result );
}
```

Параметры для таблицы стилей можно передать в хэше, как показано в следующем фрагменте:

```
# Код, не отличающийся от предыдущего примера, опущен.

my %params = {
    param1 => 10,
    param2 => 'foo',
} ;

foreach my $file ( @source_files ) {
    my $source_doc = $parser->parse_file( $file );
    my $result = $stylesheet->transform($source_doc, %params);
    print $stylesheet->output_string( $result );
}
```

Возможность передавать параметры из Perl в таблицу стилей позволяет среди прочего написать на Perl CGI-сценарий, который получает входные данные из

HTML-формы и опрашивает XML-базу данных с помощью XSLT. См. рецепт 13.5, где для той же цели мы запускали процессор XSLT в отдельном процессе вместо того, чтобы встроить его.

Еще один модуль для работы с XSLT из Perl – это `XML::Xalan`. Его автор Эдвин Пратомо (Edwin Pratomo) считает, что он все еще находится на уровне альфа-версии.

Проще всего использовать `XML::Xalan`, если таблица стилей и данные хранятся во внешних файлах:

```
use XML::Xalan;

# Конструируем объект Transformer
my $tr = new XML::Xalan::Transformer;

# Компилируем таблицу стилей
my $compiled = $tr->compile_stylesheet_file("my.xsl");

# Разбираем входной документ
my $parsed = $tr->parse_file("my.xml");

my $dest_file = "myresult.xml" ;

# Выполняем преобразование и сохраняем результат
$tr->transform_to_file($parsed, $compiled, $dest_file)
    or die $tr->errstr;
```

Иногда полезнее сохранить результат в переменной для последующей обработки:

```
my $res = $tr->transform_to_data($parsed, $compiled);
```

Предварительно разбирать входной документ и компилировать таблицу стилей необязательно, так как и то, и другое можно передать в виде файла или строки:

```
my $res = $tr->transform_to_data($src_file, $xsl_file);
```

Результатом является строка, поэтому такое применение наиболее осмысленно в случае, когда на выходе получается текст, подлежащий дальнейшей обработке средствами Perl.

Можно получить результаты и в виде событий:

```
# Создаем обработчик
$out_handler = sub {
    my ($ctx, $mesg) = @_;
    print $ctx $mesg;
};
# Вызываем преобразование, передавая обработчик
```

```
$tr->transform_to_handler(
    $xmlfile, $xslfile,
    *STDERR, $out_handler);
```

Обсуждение

Многие Perl-овики не включают в свой арсенал XSLT, поскольку человек, овладевший языком Perl, уже не понимает, как что-то можно делать по-другому. Впрочем, будем справедливы – многие программисты на Perl все же признают, что на свете есть и другие языки, а XSLT безусловно может существенно упростить сложные преобразования XML, даже если вся остальная программа написана на Perl.

См. также

В числе других решений, объединяющих Perl и XSLT, следует упомянуть модуль `XML::GNOME::XSLT` (автор T. J. Mather) – интерфейс к библиотеке libXSLT, представляющей собой процессор XSLT для проекта GNOME. Можно также попробовать реализацию XSLT на чистом Perl – модуль `XML::XSLT` (автор Jonathan Stowe). В настоящее время в ней отсутствуют такие средства XSLT 1.0, как `xsl:sort`, `xsl:key` и `xsl:import`, а ряд других реализован лишь частично. Третий вариант – модуль `XML::Sablotron` (автор Pavel Hlavnicka), представляющий собой интерфейс в написанному на C++ процессору XSLT от компании Ginger Alliance. Информацию обо всех этих модулях можно найти на сайте *http://www.cpan.org*.

Наконец, еще одна связка Perl и XSLT – комплект AxKit (*http://axkit.org*), XML-сервер приложений для Apache. В AxKit применяется конвейерная модель, позволяющая разбить обработку на несколько этапов. Функциональность XSLT основана на процессоре Sablotron.

14.17. Работа с XSLT в программах на языке Java

Задача

Требуется выполнить XSLT-преобразование в приложении, написанном на Java.

Решение

Обратиться к XSLT из программы на Java можно тремя способами:

- воспользоваться интерфейсом, который предлагает ваш любимый XSLT-процессор, написанный на Java;
- воспользоваться более переносимым интерфейсом TrAX API;
- воспользоваться пакетом JAXP 1.2 или 1.3 (надмножество TrAX, см. *http://java.sun.com/xml/jaxp/index.jsp*).

Если вы хорошо знакомы с внутренним устройством какой-нибудь конкретной Java-реализации XSLT, то возникает искушение воспользоваться предоставляемым API напрямую. Но лучше этого не делать, потому что код получится непереносимым.

Альтернативу предлагает Transformation API for XML (TrAX) – проект, который был начат по инициативе организации Apache.org (*http://xml.apache.org/xalan-j/trax.html*). Идеология, положенная в основу TrAX, лучше всего описана на сайте проекта, поэтому просто приведем цитату:

Сообщество пользователей Java выиграло бы от наличия единого API, позволяющего работать в рамках одной модели, писать код в соответствии со стандартизованными интерфейсами и применять преобразования полиморфно. Проект TrAX – это попытка определить ясную и обобщенную модель, которая в то же время удовлетворяла бы требованиям широкого спектра приложений.

Проект TrAX вошел составной частью в спецификацию JAXP 1.1 (а затем и версий 1.2 и 1.3 для J2SE 5.0), поэтому теперь есть только два способа организовать интерфейс Java и XSLT: переносимый и непереносимый. Но на вопрос, какой способ выбрать, нет однозначного ответа. У каждого процессора есть уникальные особенности, которые иногда бывают необходимы, поэтому если переносимость не имеет решающего значения, то можно и воспользоваться тем средством, которое лучше отвечает стоящей задаче. Тем не менее, в этом разделе мы будем рассматривать только переносимый JAXP 1.2 API.

С помощью JAXP 1.1 можно написать простой командный процессор XSLT, как показывает следующий пример, заимствованный из книги Eric M. Burke *Java and XSLT* (O'Reilly, 2001):

```
public class Transform
{

  public static void main(String[  ] args) throws Exception
  {
     if (args.length != 2)
     {
    System.err.println(
       "Usage: java Transform [xmlfile] [xsltfile]");
    System.exit(1);
     }

     // Открыть входной файл и таблицу стилей
     File xmlFile = new File(args[0]);
     File xsltFile = new File(args[1]);

    // Для чтения данных в JAXP используется интерфейс Source
    Source xmlSource = new StreamSource(xmlFile);
    Source xsltSource = new StreamSource(xsltFile);
```

```
        // Фабричные классы позволяют скрыть конкретный процессор XSLT
        // от приложения, так как возвращают стандартный интерфейс
        // Transformer
        TransformerFactory transFact =
           TransformerFactory.newInstance( );
        Transformer trans = transFact.newTransformer(xsltSource);

        // Применить таблицу стилей к входному документу
        trans.transform(xmlSource, new StreamResult(System.out));
    }
}
```

Помимо `StreamResult`, есть еще класс `DOMResult`, позволяющий получить результат в виде дерева DOM для последующей обработки, а также класс `SAXResult`, который дает возможность обрабатывать результаты в событийно-ориентированном стиле.

В случае DOM результат возвращается в виде объектов `Document`, `DocumentFragment` или `Element` в зависимости от типа узла, переданного конструктору `DOMResult`.

В случае `SAXResult` конструктору передается созданный пользователем объект типа `ContentHandler`, который и получает события SAX. Напомним, что SAX-обработчик определяет обратные вызовы для событий `startDocument()`, `startElement()`, `characters()`, `endElement()` и `endDocument()`. Дополнительную информацию о SAX см. сайт *http://www.saxproject.org/*.

Обсуждение

Возможность получить доступ к XSLT-преобразованиям из Java хороша не тем, что вы можете написать собственный командный интерфейс к процессору XSLT, а тем, что и без того богатейшая библиотека классов Java теперь включает и это средство.

Рассмотрим написанный на Java серверный процесс, который должен обрабатывать постоянно изменяющиеся XML-документы, хранящиеся в базе данных, или поступающие в виде SOAP-сообщений. Быть может, для обеспечения обратной совместимости сервер должен поддерживать несколько вариантов схемы документа или несколько версий SOAP-клиента. При этом обработка разных схем должна производиться прозрачно. Насколько проще был бы код сервера, если бы старую схему можно было на лету преобразовать в новую.

Приятной особенностью JAXP является тот факт, что экземпляры объектов-преобразователей можно использовать повторно, поэтому откомпилировать таблицу стилей необходимо только один раз, в момент загрузки сервера. Правда, если сервер многопоточный, и преобразования могут выполняться в каждом потоке, то для обеспечения безопасности относительно потоков придется создать несколько объектов.

См. также

В книге Eric M. Burke *Java and XSLT* (O'Reilly, 2001) приводится обширный материал по интеграции Java и XSLT с помощью JAXP 1.1. Включен исходный код нескольких готовых приложений, в том числе для организации форумов и на языке Wireless Markup Language (WML).

Глава 15. Тестирование и отладка

> Часто аварийные отказы сопровождаются сообщением вида «ID 02». ID – это сокращение от «идиосинкразия», а число обозначает, сколько еще месяцев надо тестировать продукт.
>
> *Гай Кавасаки*

15.0. Введение

Многие XSLT-сценарии пишутся для однократного применения – просто чтобы преобразовать хорошо известные входные данные. Здесь тестирование сводится к выполнению преобразования и просмотру результата. Но даже в таком простом случае как поступить с таблицей стилей, которая делает не то, что вы хотели? Обычно для отыскания ошибки достаточно внимательно проглядеть код. Но такой способ отладки не слишком эффективен, если разработчик плохо знаком с XSLT, а к таковым следует отнести и тех, кто много работает с XML, но на процедурных языках. В этой главе вы познакомитесь с основными методами отладки, помогающими быстро находить типичные ошибки, а заодно лучше разобраться в XSLT.

В этой книге я постоянно подчеркивал важность создания повторно используемого кода. Такой код автор обязан подвергнуть гораздо более тщательному тестированию. По определению, повторно используемый код часто применяется в таком контексте, о котором автор заранее не подозревал. Следует убедиться, что код работает, как описано, и для типичных входных данных, и для граничных условий. Повторно используемый код должен вести себя предсказуемо, даже если на вход поданы некорректные данные.

Разработчики более охотно тестируют программы, когда это нетрудно. Программы на интерпретируемых языках, к каковым относится и XSLT, тестировать обычно проще, поскольку отсутствует этап компиляции и сборки. Дополнительное преимущество XSLT состоит в том, что это *однородный* (homoiconic) язык, то есть сам язык и обрабатываемые им данные записаны в одном и том же синтаксисе. Это позволяет встраивать тестовые данные прямо в таблицу стилей, а, значит, создавать полностью замкнутые тесты.

Во всех рецептах из этой главы используются только стандартные средства XSLT. Но ничего нет лучше специализированного отладчика. Ниже перечислены коммерческие продукты из этой категории. Я не пробовал все из них, поэтому не считайте этот список рекомендацией. Многие продукты не ограничиваются отладкой XSLT, а умеют гораздо больше.

- Active State's Visual XSLT 2.0 (*http://www.activestate.com/Products/Visual_XSLT/*)
- Altova's XML Spy Version 2005 (*http://www.xmlspy.com/products_ide.html*)
- Emacs Based XSLT Debugger: XSLT-process (*http://xslt-process.sourceforge.net/index.php*)
- Exchanger (*http://www.exchangerxml.com/editor/index.htm*)
- MarrowSoft's Xselerator 2.6 (*http://www.marrowsoft.com/*)
- Oxygen (*http://www.oxygenxml.com/*)
- Stylus Studio 6 (*http://www.stylusstudio.com/*)
- WebSphere Studio Application Developer (*http://www-306.ibm.com/software/awdtools/studioappdev/*)

Treebeard (*http://treebeard.sourceforge.net/*) – это проект с открытыми исходными текстами. Он написан на Java, умеет работать с процессорами Xalan, Saxon, jd и Oracle XSLT. Это скорее графическая среда разработки XSLT, нежели полноценный отладчик, но помочь в отладке выражений XPath он может.

15.1. Эффективное использование xsl:message

Задача

Требуется понять, почему таблица стилей делает не то, что вы ожидаете.

Часто бывает, что вы применяете к документу таблицу стилей, а она либо ничего не выводит, либо выводит только текст без каких бы то ни было элементов. В моей практике это почти всегда означало проблему с пространствами имен. Очень легко забыть о префиксах элементов в команде `xsl:template` или допустить расхождение между URI пространства имен в документе и в таблице стилей. Напомним, что существенным является именно URI пространства имен, а не префикс. Ошибка хотя бы в одной букве приведет к неприятностям. Если URI длинный, я обычно копирую объявление пространства имен из документа в таблицу стилей, чтобы они гарантированно совпадали.

Решение

XSLT 1.0

Простейший инструмент отладки в вашем арсенале – команда `xsl:message`, которая часто применяется для того, чтобы понять, вызывался ли некий конкретный шаблон:

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

<!-- ... -->
```

```
<xsl:template match="someElement[someChild = 'someValue']">
  <xsl:message>Сопоставлен someElement[someChild = 'someValue']
</xsl:message>

    <!-- ... -->

</xsl:template>

</xsl:stylesheet>
```

Пользы от этого приема будет больше, если вывести еще и данные. Обязательно окружайте выводимый текст хорошо видными ограничителями, чтобы можно было различить информацию, выведенную разными командами `xsl:message`, а заодно понять, что выведен именно текст сообщения (а результат при этом пуст):

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

<xsl:template match="someElement[someChild = 'someValue']">
    <xsl:param name="myParam"/>
    <!-- Это неэффективный способ отладки. Если при прогоне теста вы
    ничего не увидели, это может означать лишь, что шаблон не был ни с чем
    сопоставлен или сопоставленное значение $myParam оказалось пустым -->
    <xsl:message><xsl:value-of select="$myParam"/></xsl:message>
</xsl:template>

<xsl:template match="someElement[someChild = 'someOtherValue']">
    <xsl:param name="myParam"/>
    <!-- Вот так лучше -->
    <xsl:message>Сопоставлен элемент someElement[someChild =
'someOtherValue'] </xsl:message>
    <xsl:message>$myParam=[<xsl:value-of  select="$myParam"/>]</xsl:message>
</xsl:template>

</xsl:stylesheet>
```

Предусмотрите параметр, отключающий отладку, чтобы не убирать команды `xml:message` из окончательного варианта таблицы стилей. Поместите этот параметр в отдельное пространство имен, чтобы распространяемый вами код не конфликтовал с аналогичными параметрами пользователя, который включит его в свою программу:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:dbg="http:www.ora.com/
XSLTCookbook/ns/debug">

<xsl:param name="dbg:debugOn" select="false( )"/>

<xsl:template match="someElement[someChild = 'someValue']">
```

```
    <xsl:param name="myParam"/>
    <xsl:if test="$dbg:debugOn">
      <xsl:message>Сопоставлен someElement[someChild = 'someValue']
      </xsl:message>
      <xsl:message>$myParam=[<xsl:value-of select="$myParam"/>]
      </xsl:message>
    </xsl:if>
  </xsl:template>

  </xsl:stylesheet>
```

XSLT 2.0

В версии 2.0 появилось два полезных дополнения. Во-первых, у элемента `xsl:message` теперь есть атрибут `select`, который можно использовать вместо конструктора последовательности. Во-вторых, атрибут `terminate` теперь может быть вычисляемым выражением. Это заметно упрощает жизнь, когда нужно глобально изменить поведение таблицы при завершении или создать «сообщение-утверждение» (assert).

```
<!-- Вывести значение $foo и завершить работу, если оно отрицательно -->
<xsl:message select=" 'foo=', $foo " terminate="{ if ($foo lt 0) then
'yes' else 'no'}/>
```

В некоторых процессорах XSLT реализован атрибут `use-when`, который в версии 2.0 поддерживается для всех команд. Хитрость в том, как опросить нестандартное системное свойство, применяемое во время отладки. В Saxon функция `system-property()` реализована следующим образом: если аргумент – имя в безымянном пространстве имен, то есть не имеет префикса, то считается, что это имя системного свойства Java, и возвращается значение этого свойства, если оно определено. Тем самым сообщения можно «условно компилировать».

```
<xsl:message use-when="system-property('debug_on') = 'yes' ">
    <xsl:text>Отладочный режим включен!</xsl:text>
</xsl:message>
```

Включить режим отладки можно при запуске Saxon из командной строки:

```
java -Ddebug_on=yes -jar c:\saxon\saxon8.jar test.xml test.xslt
```

Обсуждение

XSLT 1.0

Задолго до появления отладчиков существовали операторы печати. Сейчас есть несколько интерактивных отладчиков XSLT, но о бесплатных я не слыхал.

Помимо описанного выше способа использования команды `xsl:message`, ее можно применять для проверки предусловий, постусловий или инвариантов, которые должны быть истинны в некотором месте таблицы стилей:

```
<xsl:if test="debugOn>
    <xsl:if test="вставить проверку какого-то инварианта">
            <xsl:message terminate="yes">
                    Сообщение, описывающее причину сбоя.
            </xsl:message>
    </xsl:if>
</xsl:if>
```

В подобных утверждениях обычно встречается атрибут `terminate="yes"`, поскольку ошибки по определению являются фатальными.

При отладке с помощью команды `xsl:message` не следует забывать, что результат выводится в разные места в зависимости от окружения, в котором исполняется XSLT-сценарий. Например, если преобразование выполняется в браузере, то вы можете вообще не увидеть сообщений. Обычно рекомендуется протестировать таблицы стилей с помощью командного процессора и только потом исполнять их в предполагаемом окружении.

Если результат выводится в формате XML или HTML, то альтернативной `xsl:message` может служить вывод комментариев в результирующий документ с помощью команды `xsl:comment`. В частности, можно начинать каждый шаблон с комментария, чтобы протрассировать, какие шаблоны выполнялись:

```
<xsl:template match="*">
    <xsl:comment>Выведено в результате сопоставления с *</xsl:comment>
  ...
</xsl:template>

...

<xsl:template match="*" mode="foo">
    <xsl:comment>Выведено в результате сопоставления с *
    в режиме mode=foo</xsl:comment>
    ...
</xsl:template>
```

15.2. Трассировка потока обработки документа таблицей стилей

Задача

Требуется посмотреть, как таблица стилей обрабатывала документ.

Решение

XSLT 1.0

Прежде всего следует выяснить, какие возможности трассировки предоставляет используемый процессор XSLT. В Saxon есть флаг -t, который выводит

хронометраж различных стадий обработки, и флаг -T, включающий вывод трассировочной информации. В Xalan есть флаг -TT, который трассирует вызов каждого шаблона; флаг -TG трассирует каждой событие генерации; флаг -TS трассирует каждое событие отбора, а флаг -TTC – вызов дочерних шаблонов.

Если процессор вообще не поддерживает вывод трассировочной информации или вам нужен более точный контроль над выводом, то можно подумать о применении `xsl:message`. Эта команда позволяет вывести отладочную печать для трассировки выполнения таблицы стилей. Ниже приведена служебная таблица стилей, которую вы можете импортировать для этих целей:

```
<!-- xtrace.xslt -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  xmlns:dbg="http://www.ora.com/XSLTCookbook/ns/debug">

<xsl:param name="debugOn" select="false( )"/>

<xsl:template match="node( )" mode="dbg:trace" name="dbg:xtrace">
<xsl:param name="tag" select=" 'xtrace' "/>
<xsl:if test="$debugOn">
  <xsl:message>
    <xsl:value-of select="$tag"/>: <xsl:call-template name="dbg:expand-path"/>
  </xsl:message>
</xsl:if>
</xsl:template>

<!-- Продлить путь xpath до текущего узла -->
<xsl:template name="dbg:expand-path">
  <xsl:apply-templates select="." mode="dbg:expand-path"/>
</xsl:template>

<!-- Корень -->
<xsl:template match="/" mode="dbg:expand-path">
  <xsl:text>/</xsl:text>
</xsl:template>

<!-- Узел верхнего уровня -->
<xsl:template match="/*" mode="dbg:expand-path">
    <xsl:text>/</xsl:text><xsl:value-of select="name( )"/>
</xsl:template>

<!-- Узлы, имеющие родителей -->
<xsl:template match="*/*" mode="dbg:expand-path">
    <xsl:apply-templates select=".." mode="dbg:expand-path"/>/<xsl:value-of
select="name( )"/>[<xsl:number/>]<xsl:text/>
```

```
</xsl:template>

<!-- Узлы атрибутов -->
<xsl:template match="@*" mode="dbg:expand-path">
    <xsl:apply-templates select=".." mode="dbg:expand-path"/>/@<xsl:value-of
select="name( )"/>
</xsl:template>

<!-- Текстовые узлы (для ясности нормализованы) -->
<xsl:template match="text( )" mode="dbg:expand-path">normalized-
text(<xsl:value-of select="normalize-space(.)"/>)</xsl:template>

</xsl:stylesheet>
```

Включение в таблицу обращений к `dbg:xtrace` генерирует сообщения, содержащие путь к текущему узлу. В следующем примере трассируется выполнение тождественного преобразования:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                              xmlns:dbg="http://www.ora.com/
XSLTCookbook/ns/debug">

<xsl:include href="xtrace.xslt"/>

<xsl:template match="/ | node( ) | @* | comment( ) | processing-instruction( )">
    <xsl:call-template name="dbg:trace"/>
    <xsl:copy>
            <xsl:apply-templates select="@* | node( )"/>
    </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Если на вход подать такой документ:

```
<test foo="1">
    <someElement n="1"/>
    <someElement n="2">
            <someChild>someValue</someChild>
    </someElement>
    <someElement n="3">
            <someChild>someOtherValue</someChild>
    </someElement>
    <someElement n="4">
            <someChild>someValue</someChild>
    </someElement>
```

```
</test>
```

то будет выведен следующий результат:

```
xtrace: /
xtrace: /test
xtrace: /test/@foo
xtrace: normalized-text( )
xtrace: /test/someElement[1]
xtrace: /test/someElement[1]/@n
xtrace: normalized-text( )
xtrace: /test/someElement[2]
xtrace: /test/someElement[2]/@n
xtrace: normalized-text( )
xtrace: /test/someElement[2]/someChild[1]
xtrace: normalized-text(someValue)
xtrace: normalized-text( )
xtrace: normalized-text( )
xtrace: /test/someElement[3]
xtrace: /test/someElement[3]/@n
xtrace: normalized-text( )
xtrace: /test/someElement[3]/someChild[1]
xtrace: normalized-text(someOtherValue)
xtrace: normalized-text( )
xtrace: normalized-text( )
xtrace: /test/someElement[4]
xtrace: /test/someElement[4]/@n
xtrace: normalized-text( )
xtrace: /test/someElement[4]/someChild[1]
xtrace: normalized-text(someValue)
xtrace: normalized-text( )
xtrace: normalized-text( )
```

XSLT 2.0

Обращения к `dbg:trace` можно включать и выключать с помощью атрибута `use-when` точно так же, как мы поступали в случае `xsl:message` в рецепте 15.1, при условии, что процессор поддерживает опрос системных свойств.

В Saxon версии 8.поддерживаются дополнительные флаги трассировки:

```
-TJ
```

Включает трассировку связывания обращений к внешним Java-методам. Это бывает полезно, когда нужно понять, почему Saxon не находит метод, сопоставленный с функцией расширения, или выбирает конкретный метод из нескольких возможных.

```
-TL classname
```

Запускает таблицу стилей с указанным классом-получателем трассировочных сообщений `traceListener`. Аргумент `classname` – имя определенного пользователем класса, который обязан реализовать интерфейс `net.sf.saxon.trace.TraceListener`.

```
-TP
```

Запускает таблицу стилей с классом-получателем трассировочных сообщений `TimedTraceListener`. При этом создается выходной файл, в который записывается время выполнения каждой команды. Впоследствии эту информацию можно проанализировать и построить профиль выполнения таблицы стилей.

Дополнительную информацию см. в документации по Saxon.

Обсуждение

Для достижения максимального эффекта сочетайте трассировку с отладочными сообщениями, показывающими, в каком месте таблицы вы сейчас находитесь. Для этого можно использовать отдельные сообщения, но у утилиты `xtrace` есть параметр `tag`, который позволяет задать тег, печатаемый вместо стандартного.

В приведенной выше таблице *xtrace.xslt* текстовые узлы перед выводом нормализуются, чтобы трассировочное сообщение располагалось на одной строке. Если вы захотите видеть реальный текст, этот фильтр легко убрать.

Применяя трассировку, хорошенько подумайте, где разместить соответствующие вызовы. Полезнее всего ставить их в тех местах, где происходит содержательная обработка текущего узла. Рассмотрим таблицу стилей для обхода документа в обратном порядке, которую мы взяли из рецепта 5.5 и снабдили трассировкой:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dbg="http://www.ora.com/XSLTCookbook/ns/debug">

    <xsl:include href="xtrace.xslt"/>

    <xsl:output method="text"/>

    <xsl:strip-space elements="*"/>

    <xsl:template match="/employee" priority="10">
           <xsl:apply-templates/>
           <xsl:call-template name="dbg:trace"/>
           <xsl:value-of select="@name"/>
           <xsl:text> глава компания. </xsl:text>
           <xsl:call-template name="reportsTo"/>
           <xsl:call-template name="HimHer"/>
```

```
        <xsl:text>. </xsl:text>
        <xsl:text>&#xa;&#xa;</xsl:text>
    </xsl:template>

    <xsl:template match="employee[employee]">
        <xsl:apply-templates/>
        <xsl:call-template name="dbg:trace"/>
        <xsl:value-of select="@name"/>
        <xsl:text> менеджер. </xsl:text>
        <xsl:call-template name="reportsTo"/>
        <xsl:call-template name="HimHer"/>
        <xsl:text>. </xsl:text>
        <xsl:text>&#xa;&#xa;</xsl:text>
    </xsl:template>

    <xsl:template match="employee">
        <xsl:call-template name="dbg:trace"/>
        <xsl:value-of select="@name"/>
        <xsl:text> не имеет подчиненных</xsl:text>
        <xsl:text>. &#xa;</xsl:text>
    </xsl:template>

<!-- Опущено ... -->

</xsl:stylesheet>
```

Обратите внимание, что трассировка вызывается там, где выполняются какие-то действия с текущим узлом, а не просто в самом начале шаблона. При таком размещении вызовов получается следующая трасса, точно отражающая порядок обхода:

```
xtrace: /employee/employee[1]/employee[1]
xtrace: /employee/employee[1]/employee[2]/employee[1]
xtrace: /employee/employee[1]/employee[2]
xtrace: /employee/employee[1]
xtrace: /employee/employee[2]/employee[1]
xtrace: /employee/employee[2]/employee[2]/employee[1]
xtrace: /employee/employee[2]/employee[2]/employee[2]
xtrace: /employee/employee[2]/employee[2]/employee[3]
xtrace: /employee/employee[2]/employee[2]
xtrace: /employee/employee[2]
xtrace: /employee/employee[3]/employee[1]/employee[1]
xtrace: /employee/employee[3]/employee[1]/employee[2]
xtrace: /employee/employee[3]/employee[1]/employee[3]
xtrace: /employee/employee[3]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[1]/employee[1]
```

```
xtrace: /employee/employee[3]/employee[2]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[2]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[2]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[3]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[3]
xtrace: /employee/employee[3]/employee[2]
xtrace: /employee/employee[3]
xtrace: /employee
```

Если поставить обращения к трассировке в начало шаблона, то трасса будет отражать обход в прямом порядке, что лишь сбивает с толку:

```
xtrace: /employee
xtrace: /employee/employee[1]
xtrace: /employee/employee[1]/employee[1]
xtrace: /employee/employee[1]/employee[2]
xtrace: /employee/employee[1]/employee[2]/employee[1]
xtrace: /employee/employee[1]/employee[2]
xtrace: /employee/employee[1]
xtrace: /employee/employee[2]
xtrace: /employee/employee[2]/employee[1]
xtrace: /employee/employee[2]/employee[2]
xtrace: /employee/employee[2]/employee[2]/employee[1]
xtrace: /employee/employee[2]/employee[2]/employee[2]
xtrace: /employee/employee[2]/employee[2]/employee[3]
xtrace: /employee/employee[2]/employee[2]
xtrace: /employee/employee[2]
xtrace: /employee/employee[3]
xtrace: /employee/employee[3]/employee[1]
xtrace: /employee/employee[3]/employee[1]/employee[1]
xtrace: /employee/employee[3]/employee[1]/employee[2]
xtrace: /employee/employee[3]/employee[1]/employee[3]
xtrace: /employee/employee[3]/employee[1]
xtrace: /employee/employee[3]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[1]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[2]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[2]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[2]
xtrace: /employee/employee[3]/employee[2]/employee[3]
xtrace: /employee/employee[3]/employee[2]/employee[3]/employee[1]
xtrace: /employee/employee[3]/employee[2]/employee[3]
```

```
xtrace: /employee/employee[3]/employee[2]
xtrace: /employee/employee[3]
```

См. также

catchXSL! (*http://www.xslprofiler.org/overview.html*) – бесплатный инструмент для профилирования XSL-преобразований способом, зависящим от процессора. В процессе преобразования все команды XSLT протоколируются в виде *события стиля*, снабженного временным штампом. Собранная статистика дает информацию о ходе преобразования и позволяет понять, в каких местах таблицу стилей можно было бы улучшить. В подробном отчете приводится информация о каждом шаге и затраченном на него времени. Отчет в разрезе шаблонов показывает время, проведенное в каждом шаблоне, и тем самым дает возможность отыскать узкие места.

15.3. Автоматизация вставки отладочной печати

Задача

Требуется преобразовать имеющуюся таблицу стилей в другую, добавив отладочную печать.

Решение

Оливер Беккер (Oliver Becker) разработал полезное преобразование, которое принимает на входе любую таблицу стилей, а на выходе возвращает ту же таблицу, дополненную трассировкой:

```
<!--
  Утилита трассировки, модифицирует таблицу стилей,
  добавляя трассировочные сообщения Версия 0.2
  GPL (c) Oliver Becker, 2002-02-13
  obecker@informatik.hu-berlin.de
-->

<xsl:transform version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:trace="http://www.obqo.de/XSL/Trace"
  xmlns:alias="http://www.w3.org/TransformAlias"
  exclude-result-prefixes="alias">

  <xsl:namespace-alias stylesheet-prefix="alias" result-prefix="xsl" />

  <!-- <xsl:output indent="yes" /> -->

  <!-- Корневой элемент XSLT -->
  <xsl:template match="xsl:stylesheet | xsl:transform">
    <xsl:copy>
```

```
      <!-- Нам нужно пространство имен trace для имен и режимов -->
      <xsl:copy-of select="document('')/*/namespace::trace" />
      <!-- ditto: быть может, пространство имен использовалось только -->
      <!-- в качестве значения атрибута -->
      <xsl:copy-of select="namespace::*|@*" />
      <xsl:apply-templates />
      <!-- добавить служебные шаблоны -->
      <xsl:copy-of
        select="document('')/*/xsl:template
                              [@mode='trace:getCurrent' or
                              @name='trace:getPath']" />
      <!-- вычислить минимальный приоритет и добавить шаблон -->
      <!-- по умолчанию с меньшим приоритетом для узлов-элементов -->
      <xsl:variable name="priority"
              select="xsl:template/@priority
                   [not(. &gt; current( )/xsl:template/@priority)]" />
      <xsl:variable name="newpri">
        <xsl:choose>
           <xsl:when test="$priority &lt; -1">
              <xsl:value-of select="$priority - 1" />
           </xsl:when>
           <!-- на случай, если есть только более высокие приоритеты -->
          <!-- или их нет вовсе -->
           <xsl:otherwise>-2</xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <!-- копируем только содержимое -->
      <alias:template match="*" priority="{$newpri}">
        <xsl:copy-of select="document('')/*/xsl:template
                              [@name='trace:defaultRule']/node( )" />
      </alias:template>
    </xsl:copy>
</xsl:template>

<!-- шаблоны XSLT -->
<xsl:template match="xsl:template">
   <xsl:copy>
  <xsl:copy-of select="@*" />
  <!-- первое: копируем параметры -->
  <xsl:apply-templates select="xsl:param" />
  <alias:param name="trace:callstack" />
  <xsl:choose>
     <xsl:when test="@name">
        <alias:variable name="trace:current"
```

```
                select="concat($trace:callstack,'/{@name}')" />
      </xsl:when>
      <xsl:otherwise>
         <alias:variable name="trace:current"
            select="concat($trace:callstack,
                 '/{count(preceding-sibling::xsl:template)+1}')" />
        </xsl:otherwise>
      </xsl:choose>

      <!-- вывести сообщение -->
      <alias:message>
         <alias:call-template name="trace:getPath" />
         <alias:text>&#xA;   stack: </alias:text>
         <alias:value-of select="$trace:current" />
         <xsl:if test="@match or @mode">
          <alias:text> (</alias:text>
          <xsl:if test="@match">
            <alias:text>match="<xsl:value-of select="@match" />"
            </alias:text>
            <xsl:if test="@mode">
               <alias:text><xsl:text> </xsl:text></alias:text>
             </xsl:if>
          </xsl:if>
          <xsl:if test="@mode">
             alias:text>mode="<xsl:value-of select="@mode" />"</alias:text>
          </xsl:if>
          <alias:text>)</alias:text>
        </xsl:if>
        <xsl:apply-templates select="xsl:param" mode="traceParams" />
      </alias:message>

      <!-- обработать дочерние элементы, за исключением параметров -->
      <xsl:apply-templates select="node( )[not(self::xsl:param)]" />
   </xsl:copy>
</xsl:template>

<!-- добавить параметр callstack для команд apply-templates
и call-template -->
<xsl:template match="xsl:apply-templates | xsl:call-template">
   <xsl:copy>
      <xsl:copy-of select="@*" />
      <alias:with-param name="trace:callstack" select="$trace:current" />
      <xsl:apply-templates />
   </xsl:copy>
</xsl:template>
```

```
<!-- выводим значения параметров -->
<xsl:template match="xsl:param" mode="traceParams">
  <alias:text>&#xA; param: name="<xsl:value-of select="@name" />"
                    value="</alias:text>
  <alias:value-of select="${@name}" />" <alias:text />
  <!--
  <alias:copy-of select="${@name}" />" <alias:text />
  -->
</xsl:template>

<!-- выводим значения переменных -->
<xsl:template match="xsl:variable">
  <xsl:copy>
    <xsl:copy-of select="@*" />
    <xsl:apply-templates />
  </xsl:copy>
  <xsl:if test="ancestor::xsl:template">
    <alias:message> variable: name="<xsl:value-of select="@name" />"
                    value="<alias:text />
    <alias:value-of select="${@name}" />" </alias:message>
  </xsl:if>
</xsl:template>

<!-- копируем все необработанные узлы -->
<xsl:template match="*|@*">
  <xsl:copy>
    <xsl:apply-templates select="@*" />
    <xsl:apply-templates />
  </xsl:copy>
</xsl:template>

<!-- ********************************************* -->
<!-- Следующие шаблоны копируются в модифицированную -->
<!-- таблицу стилей                                -->
<!-- ********************************************* -->

<!--
 | trace:getPath
 | вычислить абсолютный путь к контекстному узлу
 +-->
<xsl:template name="trace:getPath">
  <xsl:text>node: </xsl:text>
  <xsl:for-each select="ancestor::*">
    <xsl:value-of
      select="concat('/', name( ), '[',
```

```
      count(preceding-sibling::*[name( )=name(current( ))])+1, ']')" />
  </xsl:for-each>
  <xsl:apply-templates select="." mode="trace:getCurrent" />
</xsl:template>

<!--
 | trace:getCurrent
 | вычислить последний шаг пути доступа в зависимости
 | от типа узла
 +-->
<xsl:template match="*" mode="trace:getCurrent">
  <xsl:value-of
     select="concat('/', name( ), '[',
     count(preceding-sibling::*[name( )=name(current( ))])+1, ']')" />
</xsl:template>

<xsl:template match="@*" mode="trace:getCurrent">
  <xsl:value-of select="concat('/@', name( ))" />
</xsl:template>

<xsl:template match="text( )" mode="trace:getCurrent">
  <xsl:value-of
     select="concat('/text( )[', count(preceding-sibling::text( ))+1,
                                                          ']')" />
</xsl:template>

<xsl:template match="comment( )" mode="trace:getCurrent">
  <xsl:value-of
     select="concat('/comment( )[',
                count(preceding-sibling::comment( ))+1, ']')" />
</xsl:template>

<xsl:template match="processing-instruction( )" mode="trace:getCurrent">
  <xsl:value-of
     select="concat('/processing-instruction( )[',
     count(preceding-sibling::processing-instruction( ))+1, ']')" />
</xsl:template>

<!--
 | trace:defaultRule
 | правило по умолчанию с передачей параметров
 +-->
<xsl:template name="trace:defaultRule">
  <xsl:param name="trace:callstack" />
  <xsl:message>
```

```
        <xsl:call-template name="trace:getPath" />
        <xsl:text>&#xA;   default rule applied</xsl:text>
      </xsl:message>
      <xsl:apply-templates>
        <xsl:with-param name="trace:callstack" select="$trace:callstack" />
      </xsl:apply-templates>
    </xsl:template>

</xsl:transform>
```

Обсуждение

Ниже приведен пример отладочной выдачи, получившейся в результате применения этого преобразования к таблице стилей *postorder.orgchart.xslt* из рецепта 5.5:

```
node: /employee[1]
   stack: /1 (match="/employee")
node: /employee[1]/employee[1]
   stack: /1/2 (match="employee[employee]")
node: /employee[1]/employee[1]/employee[1]
   stack: /1/2/3 (match="employee")
node: /employee[1]/employee[1]/employee[2]
   stack: /1/2/2 (match="employee[employee]")
node: /employee[1]/employee[1]/employee[2]/employee[1]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[1]/employee[2]
   stack: /1/2/2/reportsTo
node: /employee[1]/employee[1]/employee[2]
   stack: /1/2/2/HimHer
node: /employee[1]/employee[1]
   stack: /1/2/reportsTo
node: /employee[1]/employee[1]
   stack: /1/2/HimHer
node: /employee[1]/employee[2]
   stack: /1/2 (match="employee[employee]")
node: /employee[1]/employee[2]/employee[1]
   stack: /1/2/3 (match="employee")
node: /employee[1]/employee[2]/employee[2]
   stack: /1/2/2 (match="employee[employee]")
node: /employee[1]/employee[2]/employee[2]/employee[1]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[2]/employee[2]/employee[2]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[2]/employee[2]/employee[3]
```

```
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[2]/employee[2]
   stack: /1/2/2/reportsTo
node: /employee[1]/employee[2]/employee[2]
   stack: /1/2/2/HimHer
node: /employee[1]/employee[2]
   stack: /1/2/reportsTo
node: /employee[1]/employee[2]
   stack: /1/2/HimHer
node: /employee[1]/employee[3]
   stack: /1/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[1]
   stack: /1/2/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[1]/employee[1]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[1]/employee[2]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[1]/employee[3]
   stack: /1/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[1]
   stack: /1/2/2/reportsTo
node: /employee[1]/employee[3]/employee[1]
   stack: /1/2/2/HimHer
node: /employee[1]/employee[3]/employee[2]
   stack: /1/2/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[2]/employee[1]
   stack: /1/2/2/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[2]/employee[1]/employee[1]
   stack: /1/2/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[2]/employee[1]
   stack: /1/2/2/2/reportsTo
node: /employee[1]/employee[3]/employee[2]/employee[1]
   stack: /1/2/2/2/HimHer
node: /employee[1]/employee[3]/employee[2]/employee[2]
   stack: /1/2/2/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[2]/employee[2]/employee[1]
   stack: /1/2/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[2]/employee[2]/employee[2]
   stack: /1/2/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[2]/employee[2]
   stack: /1/2/2/2/reportsTo
node: /employee[1]/employee[3]/employee[2]/employee[2]
   stack: /1/2/2/2/HimHer
node: /employee[1]/employee[3]/employee[2]/employee[3]
```

```
   stack: /1/2/2/2 (match="employee[employee]")
node: /employee[1]/employee[3]/employee[2]/employee[3]/employee[1]
   stack: /1/2/2/2/3 (match="employee")
node: /employee[1]/employee[3]/employee[2]/employee[3]
   stack: /1/2/2/2/reportsTo
node: /employee[1]/employee[3]/employee[2]/employee[3]
   stack: /1/2/2/2/HimHer
node: /employee[1]/employee[3]/employee[2]
   stack: /1/2/2/reportsTo
node: /employee[1]/employee[3]/employee[2]
   stack: /1/2/2/HimHer
node: /employee[1]/employee[3]
   stack: /1/2/reportsTo
node: /employee[1]/employee[3]
   stack: /1/2/HimHer
node: /employee[1]
   stack: /1/reportsTo
node: /employee[1]
   stack: /1/HimHer
```

Модифицированная таблица стилей выводит трассировочные сообщения с помощью команды `xsl:message`. Для каждого обработанного узла печатается следующая информация:

```
node: [XPath to this node]
   stack: [состояние стека вызовов в момент обращения к данному шаблону]
   param: name="[имя параметра]" value="[значение параметра]"
   more parameters ...
   variable: name="[имя переменной]" value="[значение переменной]"
   еще переменные ...
```

Стек вызовов распечатывается в виде пути (с разделителем `/`) и включает все вызванные к этому моменту шаблоны. Если у шаблона есть атрибут `name`, то печатается его значение. В противном случае шаблон обозначается в стеке вызовов числом (позицией). Если у текущего шаблона нет имени, печатается значение атрибута `match`. Если задан атрибут `mode`, он тоже печатается.

Существует известная проблема – при обработке параметров и переменных выводится их строковое значение (результат обращения к `xsl:value-of`). Но это бессмысленно для наборов узлов и фрагментов результирующего дерева. А применение `xsl:copy-of` дает ошибку, если переменная содержит узлы атрибутов или пространств имен без родителей.

См. также

Исходный код *trace.xslt* и дополнительные примеры можно найти на странице *http://www.informatik.hu-berlin.de/~obecker/XSLT/#trace*.

15.4. Встраивание тестовых данных в служебные таблицы стилей

Задача

Требуется включить в один пакет с таблицами стилей тестовые данные для них, чтобы в любой момент можно было прогнать тесты.

Решение

Предполагается, что следующая таблица будет использоваться как служебная. Но ее можно протестировать автономно, указав в качестве тестовых входных данных ее саму:

```
<!-- math.max.xslt -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:math="http://www.exslt.org/math" exclude-result-prefixes="math"
xmlns:test="http://www.ora.com/XSLTCookbook/test" id="math:math.max">

<xsl:template name="math:max">
     <xsl:param name="nodes" select="/.."/>
     <xsl:param name="max"/>
  <xsl:variable name="count" select="count($nodes)"/>
  <xsl:variable name="aNode" select="$nodes[ceiling($count div 2)]"/>
  <xsl:choose>
     <xsl:when test="not($count)">
      <xsl:value-of select="number($max)"/>
     </xsl:when>
     <xsl:when test="number($aNode) != number($aNode)">
    <xsl:value-of select="number($aNode)"/>
      </xsl:when>
      <xsl:otherwise>
    <xsl:call-template name="math:max">
      <xsl:with-param name="nodes" select="$nodes[not(. &lt;=
number($aNode))]"/>
      <xsl:with-param name="max">
        <xsl:choose>
          <xsl:when test="not($max) or $aNode > $max">
            <xsl:value-of select="$aNode"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$max"/>
          </xsl:otherwise>
        </xsl:choose>
```

```
        </xsl:with-param>
      </xsl:call-template>
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- ТЕСТЫ: НЕ УДАЛЯТЬ! -->
<xsl:template match="/xsl:stylesheet[@id='math:math.max'] |
xsl:include[@href='math.
max.xslt'] " priority="-1000">
<xsl:message>
ТЕСТИРУЕТСЯ math.max
</xsl:message>

<xsl:for-each select="document('')/*/test:test">
  <xsl:variable name="ans">
    <xsl:call-template name="math:max">
      <xsl:with-param name="nodes" select="test:data"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:if test="$ans != @ans">
    <xsl:message>
      ТЕСТ math:max <xsl:value-of select="@num"/> НЕ ПРОШЕЛ [<xsl:value-of
select="$ans"/>]
    </xsl:message>
  </xsl:if>
</xsl:for-each>

<!-- Тест с Infinity -->
<xsl:variable name="ans1">
  <xsl:call-template name="math:max">
    <xsl:with-param name="nodes" select="document('')/*/
test:test[@num=1]/test:data"/>
    <xsl:with-param name="max" select="1 div 0"/>
  </xsl:call-template>
</xsl:variable>
<xsl:if test="$ans1 != Infinity">
  <xsl:message>
    ТЕСТ math:max с Infinity НЕ ПРОШЕЛ [<xsl:value-of select="$ans1"/>]
  </xsl:message>
</xsl:if>

<!-- Тест с -Infinity -->
<xsl:variable name="ans2">
```

```
  <xsl:call-template name="math:max">
    <xsl:with-param name="nodes" select="document('')/*/
test:test[@num=1]/test:data"/>
    <xsl:with-param name="max" select="-1 div 0"/>
  </xsl:call-template>
</xsl:variable>
<xsl:if test="$ans2 != document('')/*/test:test[@num=1]/@ans">
  <xsl:message>
    ТЕСТ math:max с -Infinity НЕ ПРОШЕЛ [<xsl:value-of select="$ans2"/>]
  </xsl:message>
</xsl:if>

</xsl:template>

<test:test num="1" ans="9" xmlns="http://www.ora.com/XSLTCookbook/test">
  <data>9</data>
  <data>8</data>
  <data>7</data>
  <data>6</data>
  <data>5</data>
  <data>4</data>
  <data>3</data>
  <data>2</data>
  <data>1</data>
</test:test>

<test:test num="2" ans="1" xmlns="http://www.ora.com/XSLTCookbook/test">
  <data>1</data>
</test:test>

<test:test num="3" ans="1" xmlns="http://www.ora.com/XSLTCookbook/test">
  <data>-1</data>
  <data>1</data>
</test:test>

<test:test num="4" ans="0" xmlns="http://www.ora.com/XSLTCookbook/test">
  <data>0</data>
  <data>0</data>
</test:test>

<test:test num="5" ans="NaN" xmlns="http://www.ora.com/XSLTCookbook/
test">
    <data>foo</data>
    <data>1</data>
```

```
</test:test>

<test:test num="6" ans="NaN" xmlns="http://www.ora.com/XSLTCookbook/test">
    <data>1</data>
    <data>foo</data>
</test:test>

<test:test num="7" ans="NaN" xmlns="http://www.ora.com/XSLTCookbook/test">
</test:test>

</xsl:stylesheet>
```

Обсуждение

У элемента `xsl:stylesheet` есть необязательный атрибут `id`. Он идентифицирует таблицы стилей, внедренные в объемлющий документ. Однако этот атрибут можно использовать и для тестирования. Наша задача – упаковать код тестирования вместе с таблицей стилей, но так, чтобы он не мешал нормальному использованию этой таблицы. Для этого создаем шаблон, который сопоставляется только тогда, когда таблица обрабатывает саму себя:

```
<xsl:template match="/xsl:stylesheet[@id='math:math.max'] |
    xsl:include[@href='math.max.xslt']">
```

Это объясняет, зачем нужна конструкция `/xsl:stylesheet[@id='math:math.max']`, но осталось непонятным, что делает часть `xsl:include[@href='math.max.xslt']`. Чтобы разобраться в этом, рассмотрим таблицу стилей, которая упаковывает все математические шаблоны в один файл, так его будет легче включать. Хотелось бы иметь простой способ и для тестирования всего пакета:

```
<!-- math.xslt -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:math="http://exslt.org/math"
    extension-element-prefixes="math" id="math:math">

<xsl:include href="math.abs.xslt"/>
<xsl:include href="math.constant.xslt"/>
<xsl:include href="math.exp.xslt"/>
<xsl:include href="math.highest.xslt"/>
<xsl:include href="math.log.xslt"/>
<xsl:include href="math.lowest.xslt"/>
<xsl:include href="math.max.xslt"/>
<xsl:include href="math.min.xslt"/>
<xsl:include href="math.power.xslt"/>
```

```
<xsl:include href="math.sqrt.xslt"/>

<!-- ТЕСТ -->
<xsl:template match="xsl:stylesheet[@id='math:math'] |
xsl:include[@href='math.xslt']">

<xsl:message>
ТЕСТИРУЕТСЯ math
</xsl:message>

    <xsl:for-each select="document('')/*/xsl:include">
           <xsl:apply-templates select="."/>
    </xsl:for-each>
</xsl:template>

<xsl:template match="xsl:include" priority="-10">
    <xsl:message>
    ПРЕДУПРЕЖДЕНИЕ: для <xsl:value-of select="@href"/> нет теста.
    </xsl:message>
</xsl:template>

</xsl:stylesheet>
```

Как видите, тестовый код для всего пакета просто перебирает элементы `xsl:include` и применяет к ним шаблоны. Благодаря наличию вышеупомянутой части `xsl:include[@href='filename']` будет произведено тестирование всех включенных таблиц стилей.

Обратите внимание на шаблон `<xsl:template match="xsl:include" priority="-10">`. Он выдает предупреждение, если для какого-то из включенных файлов нет теста. Это важно с точки зрения контроля качества, чтобы случайно не забыть включить какой-нибудь тест.

Если идея включения тестов в исполняемый код вам не по душе, то того же результата можно достичь путем создания отдельного тестового файла для каждой служебной таблицы. В таком случае необходимость в атрибуте `id` отпадает, достаточно простого сопоставления с корнем:

```
<!-- math.max.test.xslt-->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:math="http://www.exslt.org/math" exclude-result-prefixes="math"
    xmlns:test="http://www.ora.com/XSLTCookbook/test">

<xsl:include href="../math/math.max.xslt"/>

<!-- ТЕСТ: НЕ УДАЛЯТЬ! -->
```

```
<xsl:template match="/ | xsl:include[@href='math.max.test.xslt']">
<xsl:message>
ТЕСТИРУЕТСЯ math.max
</xsl:message>

<xsl:for-each select="document('')/*/test:test">
  <xsl:variable name="ans">
    <xsl:call-template name="math:max">
      <xsl:with-param name="nodes" select="test:data"/>
    </xsl:call-template>
  </xsl:variable>
  <xsl:if test="$ans != @ans">
    <xsl:message>
    math:max ТЕСТ <xsl:value-of select="@num"/> НЕ ПРОШЕЛ [<xsl:value-of
select="$ans"/>]
    </xsl:message>
  </xsl:if>
</xsl:for-each>

<!-- ... То же, что в math.max.xslt выше ... -->

</xsl:stylesheet>
```

Затем надо создать отдельные тестовые пакеты:

```
<!-- math.test.xslt -->

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:math="http://exslt.org/math"
    extension-element-prefixes="math">

<xsl:include href="math.max.test.xslt"/>
<xsl:include href="math.min.test.xslt"/>

<!-- ... То же, что в math.max.xslt выше ... -->

</xsl:stylesheet>
```

Если вы решите таким образом отделить тесты от кода, не забудьте включить их в состав дистрибутивного пакета. Тогда клиенты смогут сами прогнать тесты. Заодно тестовый код показывает, как предполагается использовать шаблоны.

15.5. Организация автономных тестов

Задача

Требуется организовать тесты, чтобы упростить процедуру тестирования.

Решение

Посмотрите, как в рецепте 15.4 мы внедрили тестовые данные в таблицу стилей. В каждом тестовом элементе имеется атрибут `num`, а правильный результат записывается в атрибут `ans`. Затем управляющая программа извлекает тестовые элементы, выполняет тест и сравнивает ожидаемый результат с получившимся. Важно то, что управляющая программа ничего не выводит, если тест прошел успешно.

Обсуждение

Ряд полезных советов по автоматизации тестирования приведен в книге Брайан Керниган, Роб Пайк *Практика программирования* (Вильямс, 2004). Авторы утверждают, что тестовые программы должны что-то выводить только, если тест не проходит. Почему? А кто захочет изучать многие страницы тестовой выдачи в поисках признаков неудачно завершившихся тестов? Если же вы знаете, что успешно выполненный тест ничего не выводит, то по наличию выдачи сразу увидите, где были ошибки. Конечно, нужно предварительно протестировать сам тестовый код и убедиться, что он действительно выполняется, иначе такая техника тестирования бесполезна.

Метод запоминания правильных ответов в атрибутах тестовых элементов работает только для простых тестов, возвращающих одиночный результат. Но ведь некоторые тесты возвращают наборы узлов. В таком случае правильный ответ можно представить в виде дочерних элементов, а затем с помощью операций над множествами из рецепта 9.2 сравнить результаты. Иногда, впрочем, протестировать шаблоны, порождающие наборы узлов, удается и более простым способом. Рассмотрим управляющую программу для тестирования шаблона `math:lowest`. Напомним, что этот шаблон возвращает набор узлов, включающий все вхождения наименьшего числа во входном наборе:

```
<xsl:stylesheet version="1.1" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" xmlns:math="http://www.exslt.org/math" exclude-result-
prefixes="math test" xmlns:test="http://www.ora.com/XSLTCookbook/test"
id="math:math.lowest">

<xsl:import href="math.min.xslt"/>

<xsl:template name="math:lowest">
   <xsl:param name="nodes" select="/.."/>

   <xsl:variable name="min">
     <xsl:call-template name="math:min">
        <xsl:with-param name="nodes" select="$nodes"/>
     </xsl:call-template>
   </xsl:variable>
   <xsl:choose>
     <xsl:when test="number($min) = $min">
        <xsl:copy-of select="$nodes[. = $min]"/>
```

```
      </xsl:when>
      <xsl:otherwise/>
    </xsl:choose>
</xsl:template>

<!-- ТЕСТ: НЕ УДАЛЯТЬ! -->
<xsl:template match="xsl:stylesheet[@id='math:math.lowest'] |
     xsl:include[@href='math.lowest.xslt'] " xmlns:exsl="http://
exslt.org/common">
   <xsl:message>
ТЕСТИРУЕТСЯ math.lowest
</xsl:message>
   <xsl:choose>
     <xsl:when test="function-available('exsl:node-set')">
      <xsl:for-each select="document('')/*/test:test">
        <xsl:variable name="ans">
         <xsl:call-template name="math:lowest">
           <xsl:with-param name="nodes" select="test:data"/>
         </xsl:call-template>
        </xsl:variable>
       <xsl:variable name="$ans-ns" select=" exsl:node-set($ans)"/>
    <xsl:if test="not($ans-ns/* != test:data[. = current( )/@ans]) and
             count($ans-ns/*) != count(test:data[. = current( )/@ans])">
        <xsl:message>
           math:lowest ТЕСТ <xsl:value-of select="@num"/> НЕ ПРОШЕЛ
           [<xsl:copy-of select="$ans-ns"/>]
           [<xsl:copy-of select="test:data[. = current( )/@ans]"/>]
        </xsl:message>
         </xsl:if>
      </xsl:for-each>
    </xsl:when>
    <xsl:otherwise>
      <xsl:message>
         ПРЕДУПРЕЖДЕНИЕ для теста math.lowest требуется exsl:node-set
        НОМЕР ВЕРСИИ=[<xsl:value-of select="system-property('xsl:version')"/>]
         ПРОИЗВОДИТЕЛЬ=[<xsl:value-of select="system-
         property('xsl:vendor')"/>]
      </xsl:message>
       </xsl:otherwise>
   </xsl:choose>
</xsl:template>

<test:test num="1" ans="1" xmlns="http://www.ora.com/XSLTCookbook/test">
   <data>9</data>
```

```
    <data>8</data>
    <data>7</data>
    <data>6</data>
    <data>5</data>
    <data>4</data>
    <data>3</data>
    <data>2</data>
    <data>1</data>
  </test:test>

  <!-- другие тесты ... >

  </xsl:stylesheet>
```

Сравнение опирается на поведение оператора !=, когда в обеих частях находятся наборы узлов: результат равен `true`, если при попарном сравнении узлов из каждого узла хотя бы у одной пары оказываются разные строковые значения. Можно проверить, что набор узлов, отобранных из тестового набора, и набор узлов, которые возвращает шаблон `math:lowest`, совпадают. Можно также убедиться, что они содержат одинаковое количество узлов.

При некоторых вычислениях (особенно приближенных математических) тест считается пройденным успешно, даже если возвращенное значение не совпадает с теоретически правильным. В таком случае можно включить в тестовые данные допустимую погрешность и убедиться, что расхождение между полученным и правильным ответом не превышает этой погрешности.

См. также

Книга Брайан Керниган, Роб Пайк *Практика программирования* (Вильямс, 2004) посвящена не конкретно XSLT, а содержит рекомендации по тестированию и отладке любых программ.

15.6. Тестирование граничных условий и ошибочных данных

Задача

Вы пишете служебные шаблоны, предназначенные для использования в чужих программах, и хотите, чтобы они были отказоустойчивыми.

Решение

Тестирование граничных условий

Во всех языках программирования ошибки чаще всего проявляются в граничных случаях. Поэтому следует выбирать тестовые данные, которые позволили

бы такие условия создать. К граничным относят минимальные и максимальные значения, да и вообще любые данные, которые лежат на границе, как бы ее ни определять в конкретном случае. Если шаблон правильно работает при таких значениях, то, скорее всего, будет работать и при всех остальных. Невозможно привести исчерпывающий перечень граничных условий, потому что в разных задачах они определяются по-разному. Но ниже мы рассмотрим некоторые типичные случаи.

Если шаблон применяется к наборам узлов (в версии 2.0 – к последовательностям), то обязательно проверяйте следующие случаи:

- пустой набор узлов;
- набор узлов, содержащий один элемент;
- набор узлов, содержащий два элемента;
- набор узлов, содержащий нечетное число элементов, большее 1;
- набор узлов, содержащий четное число элементов, большее 2.

Если шаблон применяется к строке, то проверяйте следующие случаи:

- пустая строка;
- строка длины 1;
- другие строки с разными длинами.

Если в шаблоне для поиска используются функции `substring-before` или `substring-after`, то проверяйте следующие случаи:

- строки, в которые искомая строка не входит;
- строки, которые начинаются с искомой строки;
- строки, которые заканчиваются искомой строкой;
- строка совпадает с искомой;
- строки, которые содержат несколько вхождений искомой строки (подряд и разделенные каким-то другим текстом).

Если шаблон применяется к числам, то проверяйте следующие случаи:

- число 0;
- число 1;
- отрицательные числа;
- дроби ($0 < x < 1$);
- числа, содержащие целую и дробную части;
- простые числа;
- прочие граничные значения, специфичные для конкретной задачи.

Если в шаблон сравниваются два числа X и Y, проверяйте следующие случаи:

- $X < Y$, особенно $X = Y - 1$ и $X = Y - d$, где d – небольшое дробное число;
- $X = Y$;
- $X > Y$, особенно $X = Y + 1$ и $X = Y + d$, где d – небольшое дробное число.

Если вы знаете схему подлежащего обработке документа или имеете к ней доступ, проверяйте, как поведет себя шаблон, если подать на вход данные, где:

- необязательные элементы отсутствуют;
- необязательные элементы присутствуют;

- элемент, на число экземпляров которого не наложено ограничений, встречается только один раз;
- элемент, на число экземпляров которого не наложено ограничений, встречается несколько раз;

Тестирование при ошибочных входных данных

Отказоустойчивые шаблоны и таблицы стилей должны вести себя предсказуемо при подаче на вход недопустимых данных. В таких случаях для выдачи сообщений о некорректных параметрах часто применяется команда `xsl:message` с атрибутом `terminate="yes"`.

Если шаблон применяется к числам, то проверяйте следующие случаи:

- NaN (0 div 0);
- Infinity (1 div 0);
- -Infinity (-1 div 0)
- нуль, если такое значение недопустимо (например, для логарифма);
- отрицательное значение, если оно недопустимо (например, для факториала);
- не число (например, «foo»)

Если у шаблона или таблицы стилей есть параметры, проверяйте, что происходит при следующих условиях:

- не задано значение для параметра, не имеющего значения по умолчанию;
- заданное значение выходит за пределы допустимого диапазона;
- задано значение недопустимого типа.

Проверить, что некоторый параметр не задан, можно с помощью такого приема:

```
<xsl:param name="param1">
    <xsl:message terminate="yes">
            $param1 не задан.
    </xsl:message>
</xsl:param>
```

Однако не гарантируется, что этот прием всегда будет работать, поскольку в стандарте не утверждается, что параметр вычисляется в тот момент, когда для него передано значение. Но большинство процессоров XSLT ведут себя в этом отношении «дружелюбно». Если хотите полной уверенности, то можете присвоить параметру заведомо недопустимое значение (например, 1 div 0) и проверить его в теле шаблона:

```
<xsl:param name="param1" select="1 div 0" />

<xsl:if test="$param1 = 1 div 0">
    <xsl:message terminate="yes">
       $param1 не задан или равен Infinity, что недопустимо.
    </xsl:message>
</xsl:if>
```

Если вы знаете схему подлежащего обработке документа или имеете к ней доступ, проверяйте, как поведет себя шаблон, если подать на вход документ, который[1]:

- полностью расходится со схемой (то есть на вход подан какой-то посторонний документ);
- содержит отдельные элементы, нарушающие схему;
- нарушает ограничения `minOccurs` и `maxOccurs`;
- нарушает ограничения на типы данных.

Обсуждение

Разрабатывая XSLT-код для собственных нужд, вы сами решаете, насколько отказоустойчивым он должен быть. Но если вашим кодом будут пользоваться и другие люди, то рекомендуется включать обработку ошибок (в пределах разумного). Клиенты будут только благодарны, если код работает со всеми допустимыми, пусть даже необычными, входными данными.

При создании шаблона, в котором применяется рекурсия, разумно разбить его на два. Основной шаблон проверяет ошибки и, если все нормально, вызывает шаблон с реализацией, которой и вычисляет результат рекурсивно. В рецепте 3.5 такой подход применен к вычислению логарифмов.

[1] В этом тесте предполагается, что процессор XSLT пользуется непроверяющим анализатором или вы удалили из входного документа ссылку на схему.

Глава 16. Обобщенное и функциональное программирование

Блестящие ходы, которые нам иногда доводится сделать, были бы невозможны без предыдущих глупых ходов.
Стэнли Голдстейн

16.0. Введение

Эта глава оспаривает все, что было написано в предыдущих. Ладно, это все-таки небольшое преувеличение. Речь идет о том, что во всех рассмотренных выше примерах решается та или иная конкретная задача. Полезны они и в учебных целях. Если в примере описывается не точно та задача, с которой вы столкнулись, то он все же может подсказать, в каком направлении двигаться. Возможно, достаточно будет слегка модифицировать приведенное решение или применить аналогичную технику.

В этой главе ставятся более амбициозные цели. Рассматриваемые примеры будут применимы к широкому кругу задач, причем модифицировать предложенный базовый код вообще не потребуется. Читатели, знакомые с языком C++, а в особенности со стандартной библиотекой шаблонов (STL), знают, чего можно достичь с помощью обобщенного кода (обобщенных алгоритмов), который применим в различных контекстах. А имеющие опыт работы с функциональными языками программирования (например, List, ML, Haskell) знают, какой выразительной мощью обладают функции высшего порядка: функции общего назначения, которые специализируются путем передачи других функций в качестве аргументов. В этой главе показано, что язык XSLT, хотя и не задумывался как язык для обобщенного или функционального программирования, обладает возможностями и для того, и для другого.

Техника, применяемая в этой главе, – это работа на грани возможностей языка. Не всякий захочет пользоваться этими примерами на практике, некоторые из них сложны и работают медленно. Но я вспоминаю те дни, когда в C++ еще не было встроенной поддержки шаблонов. Можно было имитировать обобщенное программирование с помощью макросов, но это выглядело так неуклюже. Однако нашлось достаточно людей, распознавших заложенный потенциал, и вскоре шаблоны стали полноценной частью языка, быть может, даже одной из важнейших его характеристик, несмотря на распространение других объектно-ориентированных языков. Подобные эксперименты оказывают давление на язык и иногда заставляют его эволюционировать быстрее. И это хорошо, потому что языки, переставшие развиваться, частенько отмирают.

Прежде чем переходить к примерам, обсудим некоторые общие приемы, которые будут применяться в этой главе. Это позволит нам в дальнейшем сконцентрировать внимание на приложении, а не на механизме.

Расширение содержимого глобальных переменных

В этой главе активно используется имеющаяся в XSLT возможность импортировать (`xsl:import`) и переопределять в импортирующей таблице шаблоны, переменные и другие элементы верхнего уровня.

Мне нравится употребляемый в объектно-ориентированных языках термин *переопределение* в применении к команде `xsl:import`. Однако технически правильно говорить, что некоторые элементы верхнего уровня в импортирующей таблице стилей имеют более высокий *приоритет импорта*, чем такие же элементы в импортируемой таблице. Подробное объяснение работы каждого элемента верхнего уровня в XSLT в части, относящейся к `xsl:import`, можно найти в книге Майкла Кэя *XSLT Programmer's Reference* (Wrox, 2001).

В этой главе будет задействована возможность объединять содержимое глобальных переменных, определенных в импортированной и в импортирующей таблицах стилей.

В следующей таблице определены две переменные. Первая – `$data1-public-data` – уникальна для данной таблицы, а вторая – `$data` – определена через первую, но ее можно переопределить:

```
<!-- data1.xslt -->

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:d="http://www.ora.com/XSLTCookbook/NS/data">

<xsl:output method="xml" indent="yes"/>

<d:data value="1"/>
<d:data value="2"/>
<d:data value="3"/>

<xsl:variable name="data1-public-data" select="document('')/*/d:*"/>
<xsl:variable name="data" select="$data1-public-data"/>

<xsl:template match="/">
    <demo>
           <xsl:copy-of select="$data"/>
    </demo>
```

```
</xsl:template>

</xsl:stylesheet>
```

Теперь создадим другую таблицу стилей, в которой значение `$data` будет расширено. В ней также определена уникальная переменная, которая объединяет открытые данные первой таблицы с локально определенными данными. Затем `$data` переопределяется в терминах объединения:

```
<!-- data2.xslt -->

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:d="http://www.ora.com/XSLTCookbook/NS/data">

<xsl:import href="data1.xslt"/>

<xsl:output method="xml" indent="yes"/>

<d:data value="5"/>
<d:data value="7"/>
<d:data value="11"/>

<xsl:variable name="data2-public-data"
    select="document('')/*/d:* | $data1-public-data"/>

<xsl:variable name="data" select="$data2-public-data"/>

</xsl:stylesheet>
```

Вот как выглядит результат работы `data1.xslt`:

```
<demo xmlns:d="data">
    <d:data value="1"/>
    <d:data value="2"/>
    <d:data value="3"/>
</demo>
```

А вот результат работы `data2.xslt`:

```
<demo xmlns:d="data">
    <d:data value="1"/>
    <d:data value="2"/>
    <d:data value="3"/>
    <d:data value="5"/>
    <d:data value="7"/>
    <d:data value="11"/>
</demo>
```

Определить переменную $data во второй таблице через ее определение в первой, не вводя дополнительных переменных, было бы удобно. Однако XSLT трактует такое определение циклически. Поэтому мы определяем именованное множество, с которым работают шаблоны, в базовой таблице стилей, но разрешаем импортирующим таблицам расширить это множество. Зачем это делается, станет ясно позже.

Метки шаблонов

В XSLT нет прямого способа передать имя одного шаблона в другой, так чтобы второй шаблон мог косвенно вызвать первый. Другими словами, следующий код недопустим ни в XSLT 1.0, ни в XSLT 2.0:

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

<!-- ТАКОЙ КОД В XSLT НЕДОПУСТИМ -->

<xsl:template match="/">
    <!-- Мы можем вызывать шаблоны по имени ...-->
    <xsl:call-template name="sayIt">
       <xsl:with-param name="aTempl" select=" 'sayHello' "/>
    </xsl:call-template>

    <xsl:call-template name="sayIt">
       <xsl:with-param name="aTempl" select=" 'sayGoodbye' "/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="sayIt">
    <xsl:param name="aTempl"/>
    <!--Но имя не разрешается задавать в виде переменной -->
    <xsl:call-template name="{$aTemple}"/>
</xsl:template>

<xsl:template name="sayHello">
    <xsl:value-of select=" 'Hello!' "/>
</xsl:template>

<xsl:template name="sayGoodbye">
    <xsl:value-of select=" 'Goodbye!' "/>
</xsl:template>

</xsl:stylesheet>
```

Оказывается, что можно было бы писать очень полезный, повторно используемый код, если бы удалось достичь такого уровня косвенности, не выходя за рамки XSLT. К счастью, это возможно, только вместо вызова по имени нужно применить

механизм сопоставления. Трюк состоит в том, чтобы определить шаблон, который может сопоставиться только с конкретным элементом данных. Этот элемент называется меткой шаблона (template tag) и, по соглашению, метка располагается непосредственно перед шаблоном, который помечает:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:f="http://www.ora.com/XSLTCookbook/namespaces/func">

<xsl:output method="text"/>

<xsl:template match="/">
    <xsl:call-template name="sayIt">
       <xsl:with-param name="aTempl" select=" 'sayHello' "/>
    </xsl:call-template>
    <xsl:call-template name="sayIt">
       <xsl:with-param name="aTempl" select=" 'sayGoodbye' "/>
    </xsl:call-template>
</xsl:template>

<xsl:template name="sayIt">
    <xsl:param name="aTempl"/>
    <!-- Применить шаблоны, указав в атрибуте select элемент-метку, -->
    <!-- уникальный для шаблона, который мы собираемся вызвать -->
    <xsl:apply-templates select="document('')/*/f:func[@name=$aTempl]"/>
</xsl:template>

<!-- Помеченный шаблон состоит из элемента-метки и шаблона, -->
<!-- сопоставляемого с этим элементом -->
<f:func name="sayHello"/>
<xsl:template match="f:func[@name='sayHello']">
    <xsl:text>Hello!&#xa;</xsl:text>
</xsl:template>

<!-- Другой помеченный шаблон -->
<f:func name="sayGoodbye"/>
<xsl:template match="f:func[@name='sayGoodbye']">
    <xsl:text>Goodbye!&#xa;</xsl:text>
</xsl:template>

</xsl:stylesheet>
```

В данном случае все эти ухищрения ни к чему, так как можно было бы просто создать шаблон, который принимает выходную строку в качестве данных. Истинная мощь этой техники раскрывается, когда помеченные функции вычисляют нечто, что может использовать вызывающий шаблон.

Применяя этот прием, не забывайте включать «сторожевой» шаблон, с которым будет производиться сопоставление, если никакой помеченный шаблон не подходит:

```
<xsl:template match="f:func">
       <xsl:message terminate="yes">
    BAD FUNC! Шаблон не соответствует объявлению generic:func.
       </xsl:message>
</xsl:template>
```

Майкл Кэй упоминает еще один способ обобщенного программирования – когда шаблон сопоставляется сам с собой:

```
<xsl:template name="f:sayHello"
           match="xsl:template[@name='f:sayHello']">
    <xsl:text>Привет!&#xa;</xsl:text>
</xsl:template>

<xsl:template name="f:sayGoodbye"
           match="xsl:template[@name='f:sayGoodbye']">
    <xsl:text>Пока!&#xa;</xsl:text>
</xsl:template>
```

Этот прием позволяет по-прежнему вызывать шаблон по имени, и выглядит шаблон вполне заурядно. В этой главе мы им пользоваться не станем, поскольку иногда будем ассоциировать с метками шаблонов дополнительные данные, так что желательно иметь два разных элемента.

ОБОБЩЕННОЕ И ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Обобщенное программирование – это способ организации повторно используемых компонентов так, чтобы их можно было легко настроить почти или совсем без потери производительности. Для использования обобщенных компонентов (классов и функций) их необходимо инстанцировать, указав конкретные типы и/или объекты.

В *функциональном программировании* используются функции высшего порядка, то есть принимающие другие функции в качестве аргументов и возвращающие функции в качестве значений.

Метод помеченных шаблонов можно интерпретировать двумя способами. С точки зрения обобщенного программирования, можно сказать, что `sayIt` – обобщенный шаблон, параметризованный другим шаблоном. А с точки зрения функционального программирования, sayIt – функция высшего порядка, принимающая другую функцию в качестве аргумента. Читатели, знакомые с обобщенным программированием в C++, наверное, скажут, что вторая интерпретация более точна, так как параметризация происходит на этапе выполнения, а не компиляции. Но я не считаю, что обобщенным программированием можно называть только то, что происходит на этапе

компиляции. Позже мы увидим, что элементы-метки могут быть не просто заместителями имени функции; в них могут быть и данные, что напоминает характеристические классы (traits), применяемые при обобщенном программировании на C++.

См. также

XSLT 1.0

Первым, кто придумал, как использовать обобщенное и функциональное программирование в XSLT, был, насколько я знаю, Димитр Новачев (Dimitre Novatchev). Он написал несколько статей на эту тему. См. *http://fxsl.sourceforge.net*. Рецепты обобщенного программирования, приведенные в этой главе, были написаны еще до того, как я познакомился с работой Новачева, и во многих отношениях наши подходы различаются. Я рекомендую знакомиться с работой Димитра только после того, как вы хорошо освоите мои примеры. Димитр заходит еще дальше, чем я, поэтому предлагаемые им методы сложнее. Он написал библиотеку FXSL для функционального программирования на языке XSLT, ее можно скачать на странице *http://sourceforge.net/project/showfiles.php?group_id=53841*.

XSLT 2.0

У Димитра Новачева есть также версия библиотеки FXSL, учитывающие новации, появившиеся в XSLT 2.0 (*http://sourceforge.net/project/showfiles.php?group_id=53841*). Самыми заметными из них являются функции и их перегрузка. Я настоятельно рекомендую эту версию всем разработчикам на XSLT 2.0, которые интересуются программированием «высшего порядка».

Ниже приводится несколько цитат из сообщения Димитра о библиотеке FSCL и функциональном программировании на XSLT. Здесь *HOF* означает Higher Order Function (функция высшего порядка).

Плюсы представляются очевидными:

1. Возможность композиции: f:map может находиться в любом месте, где могла бы быть цепочка функциональной композиции.
2. Гораздо проще и понятнее написать

```
f:map(fsomeFun( ), $seq)
```

чем

```
<xsl:for-each select="$seq">
   <xsl:sequence select="f:apply(fsomeFun( ), $seq)"/>
</xsl:for-each>
```

Наличие HOF-обертки с тем же именем и сигнатурой, что у любой функции (F) или оператора (O), дает по существу систему функционального программирования, в которой все стандартные функции и операторы XPath 2.0 (и все стандартные функции XSLT 2.0) становятся функциями высшего порядка.

Программист может сразу же начинать пользоваться всеми стандартными F & O XPath 2.0 (с которыми он хорошо знаком), ничего не делая дополнительно. На обучение времени практически не тратится. Весь богатый набор функций уже реализован и гарантированно поддерживается любым XSLT 2.0-совместимым процессором.

Еще один важный результат – это набор строгих и простых правил, описывающих, как писать пользовательские функции (`xsl:function`), чтобы они были функциями высшего порядка.

Если придерживаться этих правил, то комбинация XSLT 2.0 + FXSL становится по-настоящему замкнутой (относительно поддержки HOF) системой функционального программирования.

И еще одно преимущество – все HOF-функции потенциально можно использовать и вне XSLT (например, если заранее откомпилировать их), например, при встраивании XPath 2.0 в язык программирования и почему бы не в XQuery. Конечно, при таком подходе любая новая HOF должна быть сначала реализована на XSLT, а потом откомпилирована.

Подводя итог, можно сказать:

Система функционального программирования FXSL + XSLT 2.0 достигла такого состояния, при котором решения многих очень трудных задач можно выразить в виде однострочного выражения на языке XPath.

Это может привести к радикальным изменениям в осмыслении того, как следует решать задачи на XSLT.

Далее Димитр демонстрирует, как можно выполнить сложные математические действия с помощью FXSL:

Другая причина заключается в том, что XSLT позволяет записывать математические вычисления просто, компактно и элегантно. Например, чтобы получить последовательность:

```
N^10, N = 1 to 10
```

достаточно одной строки:

```
f:map(f:flip(f:pow( ),10), 1 to 10)
```

А вот как вычисляется сумма десятых степеней всех чисел от 1 до 10,

```
sum(f:map(f:flip(f:pow( ),10), 1 to 10))
```

И напоследок корень десятой степени из получившейся суммы:

```
f:pow(sum(f:map(f:flip(f:pow( ),10), 1 to 10)), 0.1)
```

16.1. Создание полиморфного XSLT-кода

Задача

Требуется написать XSLT-код, который выполняет одну и ту же функцию над несопоставимыми данными.

Решение

В XSLT есть два вида полиморфного поведения. Первый напоминает *перегрузку*, второй – *переопределение*.

В некоторых современных языках, особенно в C++, можно создавать перегруженные функции, то есть функции с одинаковым именем, но с разным числом или типами аргументов. Компилятор определяет, какой вариант функции вызвать, анализируя переданные ей в момент вызова параметры. В XSLT прямого аналога этого механизма нет, но давайте рассмотрим следующую таблицу стилей:

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

<xsl:output method="html" />

<xsl:template match="/">
  <html>
    <head>
      <title>Площади фигур</title>
    </head>
    <body>
      <h1>Площади фигур</h1>
      <table cellpadding="2" border="1">
        <tbody>
          <tr>
            <th>Фигура</th>
            <th>Ид фигуры</th>
            <th>Площадь</th>
          </tr>
          <xsl:apply-templates/>
        </tbody>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="shape">
  <tr>
    <td><xsl:value-of select="@kind"/></td>
    <td><xsl:value-of select="@id"/></td>
    <xsl:variable name="area">
      <xsl:apply-templates select="." mode="area"/>
    </xsl:variable>
    <td align="right"><xsl:value-of select="format-number($area,'#.000')"/>
</td>
  </tr>
```

```
</xsl:template>

<xsl:template match="shape[@kind='triangle']" mode="area">
  <xsl:value-of select="@base * @height"/>
</xsl:template>

<xsl:template match="shape[@kind='square']" mode="area">
  <xsl:value-of select="@side * @side"/>
</xsl:template>

<xsl:template match="shape[@kind='rectangle']" mode="area">
  <xsl:value-of select="@width * @height"/>
</xsl:template>

<xsl:template match="shape[@kind='circle']" mode="area">
  <xsl:value-of select="3.1415 * @radius * @radius"/>
</xsl:template>

</xsl:stylesheet>
```

Обратите внимание, что у нескольких шаблонов есть режим `area`. Эти шаблоны принимают разные типы фигур и по-разному вычисляют их площадь. Если вместо слова *mode* (режим) подставить *имя функции*, а вместо *образец для сопоставления – тип данных*, то сразу станет понятно, что перед нами вариант полиморфной перегрузки.

Второй вид полиморфизма – переопределение. В этой книги вы видели многочисленные примеры переопределения. Для реализации такого поведения в XSLT используется команда `xsl:import`. Приведенный ниже пример – это модификация таблицы стилей для DocBook из примера 10.1. Чтобы сделать ее расширяемой, мы внесли следующие изменения:

- для определения примитивных компонентов содержимого атрибутов используются переменные. Эти переменные можно переопределить в импортирующей таблице стилей;
- используются наборы атрибутов, в которые импортирующая таблица может добавить новые атрибуты или переопределить существующие;
- для каждого раздела документа используется простой шаблон, который можно переопределить в импортирующей таблице стилей;
- введена *точка вклинивания* – вызов именованного шаблона `extra-head-meta-data`; реализация по умолчанию не делает ничего.

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

  <!-- Переменные, определяющие различные компоненты стилей -->
  <xsl:variable name="standard-font-family" select=" 'font-family:
```

```
  Times serif; font-weight' "/>

 <xsl:variable name="chapter-label-font-size" select=" 'font-size : 48pt' "/>
 <xsl:variable name="chapter-title-font-size" select=" 'font-size : 24pt' "/>
  <xsl:variable name="epigraph-font-size" select=" 'font-size : 10pt' "/>
  <xsl:variable name="sect1-font-size" select=" 'font-size : 18pt' "/>
  <xsl:variable name="sect2-font-size" select=" 'font-size : 14pt' "/>
  <xsl:variable name="normal-font-size" select=" 'font-size : 12pt' "/>

  <xsl:variable name="normal-text-color" select=" 'color: black' "/>
  <xsl:variable name="chapter-title-color" select=" 'color: red' "/>

  <xsl:variable name="epigraph-padding" select=" 'padding-top:4;
  padding-bottom:4' "/>

  <xsl:variable name="epigraph-common-style" select="concat($standard-
font-family,'; ', $epigraph-font-size, '; ', $epigraph-padding, '; ',
$normal-text-color)"/>
  <xsl:variable name="sect-common-style" select="concat($standard-font-
family,'; font-weight: bold', '; ',$normal-text-color)"/>

  <!-- Наборы атрибутов -->
  <xsl:attribute-set name="chapter-align">
    <xsl:attribute name="align">right</xsl:attribute>
  </xsl:attribute-set>

  <xsl:attribute-set name="normal-align">
  </xsl:attribute-set>

  <xsl:attribute-set name="chapter-label" use-attribute-sets="chapter-align">
    <xsl:attribute name="style">
      <xsl:value-of select="$standard-font-family"/>;
      <xsl:value-of select="$chapter-label-font-size"/>;
      <xsl:value-of select="$chapter-title-color"/>
      <xsl:text>; padding-bottom:10; font-weight: bold</xsl:text>
    </xsl:attribute>
  </xsl:attribute-set>

  <xsl:attribute-set name="chapter-title" use-attribute-sets="chapter-align">
    <xsl:attribute name="style">
      <xsl:value-of select="$standard-font-family"/>;
      <xsl:value-of select="$chapter-title-font-size"/>;
      <xsl:value-of select="$chapter-title-color"/>
      <xsl:text>; padding-bottom:150; font-weight: bold</xsl:text>
```

```
  </xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="epigraph-para" use-attribute-sets="chapter-align">
  <xsl:attribute name="style">
    <xsl:value-of select="$epigraph-common-style"/><xsl:text>;
    font-style: italic</xsl:text>
  </xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="epigraph-attribution"
use-attribute-sets="chapter-align">
  <xsl:attribute name="style">
    <xsl:value-of select="$epigraph-common-style"/>
  </xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="sect1">
  <xsl:attribute name="align">left</xsl:attribute>
  <xsl:attribute name="style">
    <xsl:value-of select="$sect-common-style"/>;
    <xsl:value-of select="$sect1-font-size"/>
  </xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="sect2">
  <xsl:attribute name="align">left</xsl:attribute>
  <xsl:attribute name="style">
    <xsl:value-of select="$sect-common-style"/>;
    <xsl:value-of select="$sect2-font-size"/>
  </xsl:attribute>
</xsl:attribute-set>

<xsl:attribute-set name="normal">
  <xsl:attribute name="align">left</xsl:attribute>
  <xsl:attribute name="style">
    <xsl:value-of select="$standard-font-family"/>;
    <xsl:value-of select="$normal-font-size"/>;
    <xsl:value-of select="$normal-text-color"/>
  </xsl:attribute>
</xsl:attribute-set>

<!-- Шаблоны -->
<xsl:template match="/">
  <html>
```

```
      <head>
           <xsl:apply-templates mode="head"/>
           <xsl:call-template name="extra-head-meta-data"/>
      </head>
      <body style=
      "margin-left:100;margin-right:100;margin-top:50;margin-bottom:50">
        <xsl:apply-templates/>
        <xsl:apply-templates select="chapter/chapterinfo/*"
mode="copyright"/>
      </body>
    </html>

  </xsl:template>

  <!-- Заголовок -->

  <xsl:template match="chapter/title" mode="head">
        <title><xsl:value-of select="."/></title>
  </xsl:template>

  <xsl:template match="author" mode="head">
        <meta name="author" content="{concat(firstname,' ', surname)}"/>
  </xsl:template>

  <xsl:template match="copyright" mode="head">
        <meta name="copyright" content="{concat(holder,' ',year)}"/>
  </xsl:template>

  <xsl:template match="text( )" mode="head"/>

  <!-- Переопределить в импортирующей таблице стилей, если необходимо -->
  <xsl:template name="extra-head-meta-data"/>

<!-- Тело -->

  <xsl:template match="chapter">
     <div xsl:use-attribute-sets="chapter-label">
        <xsl:value-of select="@label"/>
    </div>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="chapter/title">
    <div xsl:use-attribute-sets="chapter-title">
      <xsl:value-of select="."/>
```

```
    </div>
  </xsl:template>

  <xsl:template match="epigraph/para">
    <div xsl:use-attribute-sets="epigraph-para">
      <xsl:value-of select="."/>
    </div>
  </xsl:template>

  <xsl:template match="epigraph/attribution">
    <div xsl:use-attribute-sets="epigraph-attribution">
      <xsl:value-of select="."/>
    </div>
  </xsl:template>

  <xsl:template match="sect1">
    <h1 xsl:use-attribute-sets="sect1">
      <xsl:value-of select="title"/>
    </h1>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="sect2">
    <h2 xsl:use-attribute-sets="sect2">
    <xsl:value-of select="title"/>
    </h2>
     <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="para">
    <p xsl:use-attribute-sets="normal">
      <xsl:value-of select="."/>
    </p>
  </xsl:template>

  <xsl:template match="text( )"/>

<xsl:template match="copyright" mode="copyright">
  <div style="font-size : 10pt; font-family: Times serif; padding-top : 100">
    <xsl:text>Copyright </xsl:text>
    <xsl:value-of select="holder"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="year"/>
    <xsl:text>. All rights reserved.</xsl:text>
```

```
  </div>
</xsl:template>

<xsl:template match="*" mode="copyright"/>

</xsl:stylesheet>
```

Обсуждение

Назвать XSLT объектно-ориентированным языком значило бы приукрасить истину. Поведение `xsl:import` лишь отдаленно напоминает наследование и работает эта команда на уровне таблицы в целом. Ни о какой инкапсуляции или абстрагировании данных и речи не идет. Тем не менее, программисты с объектно-ориентированным складом ума могут применить накопленный опыт для создания таблиц стилей с большей степенью модульности и повторной используемости.

Взгляните еще раз на пример перегрузки, приведенный в разделе «Решение». Всякий раз, как возникает необходимость выполнять схожие операции над несопоставимыми данными, имеет смысл именно так структурировать таблицу стилей. Конечно, желаемого результата можно достичь и с помощью условий:

```
<xsl:template match="shape">
  <tr>
    <td><xsl:value-of select="@kind"/></td>
    <td><xsl:value-of select="@id"/></td>
    <xsl:variable name="area">
      <xsl:call-template name="area"/>
    </xsl:variable>
    <td align="right"><xsl:value-of select="format-number($area,'#.000')"/>
    </td>
  </tr>
</xsl:template>

<xsl:template name="area">
  <xsl:choose>

    <xsl:when test="@kind='triangle' ">
      <xsl:value-of select="@base * @height"/>
    </xsl:when>

    <xsl:when test="@kind='square' " >
      <xsl:value-of select="@side * @side"/>
    </xsl:when>

    <xsl:when test="@kind='rectangle' ">
```

```
      <xsl:value-of select="@width * @height"/>
    </xsl:when>

    <xsl:when test="@kind='circle' ">
      <xsl:value-of select="3.1415 * @radius * @radius"/>
    </xsl:when>

  </xsl:choose>

</xsl:template>
```

На таком простом примере трудно аргументировать превосходство перегрузки над условной логикой. Но представьте, что кто-то захотел воспользоваться этой таблицей стилей, но треугольники задавать с помощью длин двух сторон и угла между ними. Если в таблице есть отдельные шаблоны для каждой фигуры, то импортировать ее и расширить поведение легко. Если же логика вычисления площади представлена в виде монолитного набора условий, то пользователю придется скопировать всю таблицу и изменить некоторые части или добавить новые условия, рискуя ненароком внести ошибку.

Переопределение – ключ к созданию модульного повторно используемого XSLT-кода. Однако повторная используемость обходится не даром, требуется заранее все спланировать. Конкретно, нужно обдумать три вещи:

1. Что могут захотеть изменить пользователи вашей таблицы стилей?
2. Что, на ваш взгляд, они не должны изменять?
3. Что они, скорее всего, изменять не будут?

Если не ответить на первый вопрос, то таблица стилей окажется недостаточно гибкой и использовать ее повторно можно будет только путем копирования, вставки и изменения. Не ответив на второй вопрос, вы рискуете расставить капканы для пользователей. А не задумавшись над третьим вопросом, вы создадите излишне сложную таблицу стилей, которая будет работать медленно и окажется неудобной для сопровождения.

В таблице стилей для DocBook мы реализовали удобный механизм переопределения шрифтов и других атрибутов текста по отдельности. Переопределять атрибуты выравнивания уже не так удобно, поскольку в результате могло бы получиться несогласованное выравнивание фрагментов текста, которые должны быть выровнены одинаково и определенным образом. Например, изменить выравнивание раздела и обычного текста, задавая атрибуты порознь, гораздо труднее. Наконец, мы вообще не стали облегчать задачу переопределения имен конкретных HTML-элементов, так как это резко усложнило бы таблицу стилей, не давая ощутимых преимуществ.

См. также

Крис Ратман (Chris Rathman) приводит гораздо более обширный пример полиморфного XSLT-кода на странице *http://www.angelfire.com/tx4/cus/shapes/xsl.html*.

16.2. Создание обобщенных функций агрегирования элементов

Задача

Требуется создать повторно используемые шаблоны для выполнения широкого спектра операций агрегирования набора узлов.

Решение

В обобщенном расширяемом решении применяется метод пометки узлов, описанный во введении к этой главе.

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:aggr="http://www.ora.com/XSLTCookbook/namespaces/aggregate"
  extension-element-prefixes="generic">

  <xsl:variable name="generic:public-generics"
                select="document('')/*/generic:*"/>

   <xsl:variable name="generic:generics" select="$generic:public-generics"/>

  <!-- Примитивные обобщенные функции от x -->

<generic:func name="identity"/>
  <xsl:template match="generic:func[@name='identity']">
       <xsl:param name="x"/>
       <xsl:value-of select="$x"/>
  </xsl:template>

  <generic:func name="square"/>
  <xsl:template match="generic:func[@name='square']">
       <xsl:param name="x"/>
       <xsl:value-of select="$x * $x"/>
  </xsl:template>

  <generic:func name="cube"/>
  <xsl:template match="generic:func[@name='cube']">
       <xsl:param name="x"/>
       <xsl:value-of select="$x * $x * $x"/>
  </xsl:template>

  <generic:func name="incr" param1="1"/>
```

```
  <xsl:template match="generic:func[@name='incr']">
       <xsl:param name="x"/>
       <xsl:param name="param1" select="@param1"/>
       <xsl:value-of select="$x + $param1"/>
  </xsl:template>

<!-- Примитивные обобщенные агрегаторы -->

  <generic:aggr-func name="sum" identity="0"/>
  <xsl:template match="generic:aggr-func[@name='sum']">
       <xsl:param name="x"/>
       <xsl:param name="accum"/>
       <xsl:value-of select="$x + $accum"/>
  </xsl:template>

  <generic:aggr-func name="product" identity="1"/>
  <xsl:template match="generic:aggr-func[@name='product']">
       <xsl:param name="x"/>
       <xsl:param name="accum"/>
       <xsl:value-of select="$x * $accum"/>
  </xsl:template>

  <!-- Обобщенный шаблон агрегирования -->
  <xsl:template name="generic:aggregation">
    <xsl:param name="nodes"/>
    <xsl:param name="aggr-func" select=" 'sum' "/>
    <xsl:param name="func" select=" 'identity' "/>
    <xsl:param name="func-param1"
               select="$generic:generics[self::generic:func and
                                         @name = $func]/@param1"/>
    <xsl:param name="i" select="1"/>
    <xsl:param name="accum"
               select="$generic:generics[self::generic:aggr-func and
                                        @name = $aggr-func]/@identity"/>
     <xsl:choose>
      <xsl:when test="$nodes">

       <!-- Вычислить func($x) -->
        <xsl:variable name="f-of-x">
          <xsl:apply-templates
                   select="$generic:generics[self::generic:func and
                                             @name = $func]">
            <xsl:with-param name="x" select="$nodes[1]"/>
            <xsl:with-param name="i" select="$i"/>
```

```
            <xsl:with-param name="param1" select="$func-param1"/>
          </xsl:apply-templates>
        </xsl:variable>

        <!-- Агрегировать текущее значение $f-of-x с $accum -->
        <xsl:variable name="temp">
          <xsl:apply-templates
              select="$generic:generics[self::generic:aggr-func and
                              @name = $aggr-func]">
            <xsl:with-param name="x" select="$f-of-x"/>
            <xsl:with-param name="accum" select="$accum"/>
            <xsl:with-param name="i" select="$i"/>
          </xsl:apply-templates>
        </xsl:variable>

        <!-- Хвостовая рекурсия для обработки остальных узлов
           с использованием position( ) -->
        <xsl:call-template name="generic:aggregation">
          <xsl:with-param name="nodes" select="$nodes[position( )!=1]"/>
          <xsl:with-param name="aggr-func" select="$aggr-func"/>
          <xsl:with-param name="func" select="$func"/>
          <xsl:with-param name="func-param1" select="$func-param1"/>
          <xsl:with-param name="i" select="$i + 1"/>
          <xsl:with-param name="accum" select="$temp"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$accum"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

Обобщенный код состоит из трех основных частей.

Первая часть – это помеченные обобщенные функции от одной переменной `x`. Они позволяют выполнять операции агрегирования, применяя функции к узлам из входного набора. Простейшей из них является функция `identity`, применяемая, когда нужно агрегировать сами значения входных узлов. Также определены функции для возведения в квадрат, в куб и для увеличения на заданную величину.

Вторая часть – помеченные обобщенные функции-агрегаторы. Мы реализовали два типичных агрегатора: сумма и произведение. Импортирующая таблица может добавить и другие.

Третья часть – обобщенный алгоритм агрегирования. В качестве параметров он принимает набор агрегируемых узлов, имя функции-агрегатора (по умолчанию `sum`) и имя функции, применяемой к одному узлу (по умолчанию `identity`). Параметр `$i` нужен для отслеживания позиции текущего обрабатываемого узла; он передается и агрегатору, и функции обработки узла на случай, если вдруг понадобится. В параметре `$accum` хранится текущее значение агрегата. Отметим, что по умолчанию этот параметр инициализируется значением атрибута `@identity`, который хранится в метке функции-агрегатора. Такая инициализация демонстрирует присущую обобщенному подходу возможность ассоциировать метаданные с метками. Это отдаленно напоминает классы-характеристики, часто применяемые при обобщенном программировании на C++.

Чтобы разобраться в этом коде, мы начнем с простого приложения, которое пользуется готовыми средствами агрегирования и расширяет их. См. пример 16.1.

Пример 16.1. Использование и расширение обобщенного агрегирования

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:aggr="http://www.ora.com/XSLTCookbook/namespaces/aggregate"
  extension-element-prefixes="generic aggr">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширить набор обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Добавить обобщенную функцию для вычисления обратного значения -->
<generic:func name="reciprocal"/>
<xsl:template match="generic:func[@name='reciprocal']">
     <xsl:param name="x"/>
     <xsl:value-of select="1 div $x"/>
</xsl:template>

<!-- Добавить обобщенный агрегатор для вычисления минимального --Ю
<!-- и максимального узла в наборе -->
<generic:aggr-func name="min" identity=""/>
<xsl:template match="generic:aggr-func[@name='min']">
     <xsl:param name="x"/>
     <xsl:param name="accum"/>
  <xsl:choose>
    <xsl:when test="$accum = @identity or $accum >= $x">
```

```
      <xsl:value-of select="$x"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$accum"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<generic:aggr-func name="max" identity=""/>
<xsl:template match="generic:aggr-func[@name='max']">
     <xsl:param name="x"/>
     <xsl:param name="accum"/>
  <xsl:choose>
    <xsl:when test="$accum = @identity or $accum &lt; $x">
      <xsl:value-of select="$x"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$accum"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Тестируем функциональность агрегирования -->
<xsl:template match="numbers">

<results>

  <!-- Сумма чисел -->
  <sum>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="number"/>
    </xsl:call-template>
 </sum>

  <!-- Сумма квадратов -->
  <sumSq>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'square' "/>
    </xsl:call-template>
  </sumSq>

  <!-- Произведение обратных значений -->
  <prodRecip>
```

```
      <xsl:call-template name="generic:aggregation">
        <xsl:with-param name="nodes" select="number"/>
        <xsl:with-param name="aggr-func" select=" 'product' "/>
        <xsl:with-param name="func" select=" 'reciprocal' "/>
      </xsl:call-template>
    </prodRecip>

    <!-- Максимум -->
    <max>
        <xsl:call-template name="generic:aggregation">
        <xsl:with-param name="nodes" select="number"/>
        <xsl:with-param name="aggr-func" select=" 'max' "/>
      </xsl:call-template>
    </max>

    <!-- Минимум -->
    <min>
        <xsl:call-template name="generic:aggregation">
        <xsl:with-param name="nodes" select="number"/>
        <xsl:with-param name="aggr-func" select=" 'min' "/>
      </xsl:call-template>
    </min>

  </results>

  </xsl:template>

  </xsl:stylesheet>
```

В примере 16.1 показано, как добавлять новые функции, применяемые к элементу, и функции-агрегаторы к тем, что уже включены в таблицу *aggregation.xslt*. Быть может, вы не думали, что минимум и максимум можно вычислить с помощью этого обобщенного кода, однако, как видите, это совсем просто.

Протестируем код на следующих входных данных:

```
<numbers>
    <number>1</number>
    <number>2</number>
    <number>3</number>
</numbers>
```

В результате получаем:

```
<?xml version="1.0" encoding="utf-8"?>
<results>
    <sum>6</sum>
```

```
    <sumSq>14</sumSq>
    <prodRecip>0.1666666666666666</prodRecip>
    <max>3</max>
    <min>1</min>
</results>
```

Обсуждение

Приведенный в разделе «Решении» пример – это лишь верхушка айсберга. С помощью обобщенного агрегирования можно делать и куда более интересные вещи. Например, нигде не сказано, что агрегировать можно только числа. Ниже показано, что этот обобщенный код применим также и к строкам:

```
<strings>
  <string>camel</string>
  <string>through</string>
  <string>the</string>
  <string>eye</string>
  <string>of</string>
  <string>needle</string>
</strings>

<!DOCTYPE stylesheet [
     <!ENTITY % standard SYSTEM "../strings/standard.ent">
     %standard;
]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  extension-element-prefixes="generic">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширить набор обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Добавить обобщенную функцию для преобразования первого символа $x
в верхний регистр -->
<generic:func name="upperFirst"/>
<xsl:template match="generic:func[@name='upperFirst']">
     <xsl:param name="x"/>
   <!-- О том, что такое LOWER_TO_UPPER, см. рецепт 2.8 -->
```

```
      <xsl:variable name="upper"
                    select="translate(substring($x,1,1),&LOWER_TO_UPPER;)"/>
      <xsl:value-of select="concat($upper, substring($x,2))"/>
</xsl:template>

<!-- Добавить обощенный агрегатор для конкатенации строк -->
<generic:aggr-func name="concat" identity=""/>
<xsl:template match="generic:aggr-func[@name='concat']">
      <xsl:param name="x"/>
      <xsl:param name="accum"/>
      <xsl:value-of select="concat($accum,$x)"/>
</xsl:template>

<!-- Тестируем функциональность агрегирования -->
<xsl:template match="strings">

<results>

  <camelCase>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="string"/>
      <xsl:with-param name="aggr-func" select=" 'concat' "/>
      <xsl:with-param name="func" select=" 'upperFirst' "/>
    </xsl:call-template>
 </camelCase>

</results>

</xsl:template>

</xsl:stylesheet>

<results>
   <camelCase>CamelThroughTheEyeOfNeedle</camelCase>
</results>
```

С помощью агрегирования можно вычислить также и статистические функции: среднее и дисперсию. Нам понадобится параметр `$i`. Для вычисления дисперсии придется проявить изобретательность; в параметре `$accum` необходимо хранить три величины: сумму, сумму квадратов и текущее значение агрегата. Сделать это позволит элемент с атрибутом. Единственный недостаток этого решения – необходимость использовать функцию `node-set()` в XSLT 1.0:

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

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:exslt="http://exslt.org/common"
  extension-element-prefixes="generic exslt">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширить набор обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Добавить обощенный агрегатор для вычисления дисперсии -->
<generic:aggr-func name="avg" identity="0"/>
<xsl:template match="generic:aggr-func[@name='avg']">
     <xsl:param name="x"/>
     <xsl:param name="accum"/>
     <xsl:param name="i"/>
     <xsl:value-of select="(($i - 1) * $accum + $x) div $i"/>
</xsl:template>

<generic:aggr-func name="variance" identity=""/>
<xsl:template match="generic:aggr-func[@name='variance']">
  <xsl:param name="x"/>
  <xsl:param name="accum"/>
  <xsl:param name="i"/>

  <xsl:choose>
    <xsl:when test="$accum = @identity">
      <!-- Инициализировать, сумму, сумму квадратов
           и дисперсию. Дисперсия одного числа равна 0. -->
      <variance sum="{$x}" sumSq="{$x * $x}">0</variance>
    </xsl:when>
    <xsl:otherwise>
      <!-- Используем node-set для преобразования $accum
          в набор узлов, содержащий элемент, представляющий дисперсию. -->
      <xsl:variable name="accumElem" select="exslt:node-set($accum)/*"/>
      <!-- Добавить к сумме значение $x -->
      <xsl:variable name="sum" select="$accumElem/@sum + $x"/>
      <!-- Добавить к сумме квадрат значения $x -->
      <xsl:variable name="sumSq" select="$accumElem/@sumSq + $x * $x"/>
      <!-- Возвращаем элемент, у которого есть атрибуты для
       представления суммы и суммы квадратов, а текущая величина
```

```
    дисперсии хранится в виде значения элемента. -->
      <variance sum="{$sum}" sumSq="{$sumSq}">
        <xsl:value-of
           select="($sumSq - ($sum * $sum) div $i) div ($i - 1)"/>
      </variance>
    </xsl:otherwise>
  </xsl:choose>

</xsl:template>

<xsl:template match="numbers">

<results>

  <!-- Среднее -->
  <avg>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="aggr-func" select=" 'avg' "/>
    </xsl:call-template>
 </avg>

  <!-- Среднее квадратов -->
  <avgSq>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'square' "/>
      <xsl:with-param name="aggr-func" select=" 'avg' "/>
    </xsl:call-template>
  </avgSq>

  <!-- Дисперсия -->
  <variance>
    <xsl:call-template name="generic:aggregation">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="aggr-func" select=" 'variance' "/>
    </xsl:call-template>
  </variance>

</results>

</xsl:template>

</xsl:stylesheet>
```

В следующем примере показано, как применить агрегирование для вычисления суммы полиморфных функций:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/
generic" xmlns:aggr="http://www.ora.com/XSLTCookbook/namespaces/
aggregate" xmlns:exslt="http://exslt.org/common" extension-element-
prefixes="generic aggr">

  <xsl:import href="aggregation.xslt"/>

  <xsl:output method="xml" indent="yes"/>

  <!-- Расширить набор обобщенных функций -->
  <xsl:variable name="generic:generics"
      select="$generic:public-generics | document('')/*/generic:*"/>

  <!-- Расширить примитивы для вычисления комиссии -->
  <generic:func name="commission"/>
  <xsl:template match="generic:func[@name='commission']">
    <xsl:param name="x"/>
    <!-- отложить фактическое вычисление полиморфного шаблона с помощью -->
    <!-- режима commission -->
    <xsl:apply-templates select="$x" mode="commission"/>
  </xsl:template>

  <!-- По умолчанию продавец получает 2% комиссионных без основного
оклада -->
  <xsl:template match="salesperson" mode="commission">
    <xsl:value-of select="0.02 * sum(product/@totalSales)"/>
  </xsl:template>

  <!-- продавцы ранга > 4 получают оклад $10000.00 + 0.5% комиссионных -->
  <xsl:template match="salesperson[@seniority > 4]" mode="commission"
priority="1">
    <xsl:value-of select="10000.00 + 0.05 * sum(product/@totalSales)"/>
  </xsl:template>

  <!-- продавцы ранга > 8 получают оклад, равный  (ранг * $10000.00) + 0.8%
    комиссионных -->
  <xsl:template match="salesperson[@seniority > 8]" mode="commission"
priority="2">
    <xsl:value-of select="@seniority * 2000.00 + 0.08 *
          sum(product/@totalSales)"/>
```

```
</xsl:template>

<xsl:template match="salesBySalesperson">
  <results>
    <result>
      <xsl:text>Общая сумма комиссионных = </xsl:text>
      <xsl:call-template name="generic:aggregation">
         <xsl:with-param name="nodes" select="*"/>
         <xsl:with-param name="aggr-func" select=" 'sum' "/>
         <xsl:with-param name="func" select=" 'commission' "/>
      </xsl:call-template>
    </result>
    <result>
      <xsl:text>Минимальная комиссия = </xsl:text>
      <xsl:call-template name="generic:aggregation">
         <xsl:with-param name="nodes" select="*"/>
         <xsl:with-param name="aggr-func" select=" 'min' "/>
         <xsl:with-param name="func" select=" 'commission' "/>
      </xsl:call-template>
    </result>
    <result>
      <xsl:text>Максимальная комиссия = </xsl:text>
      <xsl:call-template name="generic:aggregation">
         <xsl:with-param name="nodes" select="*"/>
         <xsl:with-param name="aggr-func" select=" 'max' "/>
         <xsl:with-param name="func" select=" 'commission' "/>
      </xsl:call-template>
    </result>
    <result>
      <xsl:text>Средняя комиссия = </xsl:text>
      <xsl:call-template name="generic:aggregation">
         <xsl:with-param name="nodes" select="*"/>
         <xsl:with-param name="aggr-func" select=" 'avg' "/>
         <xsl:with-param name="func" select=" 'commission' "/>
      </xsl:call-template>
    </result>
    <result>
      <xsl:text>Средняя сумма продаж = </xsl:text>
      <xsl:call-template name="generic:aggregation">
          <xsl:with-param name="nodes" select="*/product/@totalSales"/>
          <xsl:with-param name="aggr-func" select=" 'avg' "/>
      </xsl:call-template>
    </result>
    <result>
```

```
        <xsl:text>Минимальная сумма продаж = </xsl:text>
        <xsl:call-template name="generic:aggregation">
            <xsl:with-param name="nodes" select="*/product/@totalSales"/>
            <xsl:with-param name="aggr-func" select=" 'min' "/>
        </xsl:call-template>
      </result>
      <result>
        <xsl:text>Максимальная сумма продаж = </xsl:text>
        <xsl:call-template name="generic:aggregation">
            <xsl:with-param name="nodes" select="*/product/@totalSales"/>
            <xsl:with-param name="aggr-func" select=" 'max' "/>
        </xsl:call-template>
      </result>
    </results>
  </xsl:template>

</xsl:stylesheet>
```

Применив эту таблицу стилей к файлу *salesBySalesperson.xml* (см. главу 4), получим:

```
<results xmlns:exslt="http://exslt.org/common">
   <result>Общая сумма комиссионных = 471315</result>
   <result>Минимальная комиссия = 19600</result>
   <result>Максимальная комиссия = 364440</result>
   <result>Средняя комиссия = 117828.75</result>
   <result>Средняя сумма продаж = 584636.3636363636</result>
   <result>Минимальная сумма продаж = 5500.00</result>
   <result>Максимальная сумма продаж = 2920000.00</result>
</results>
```

В этом разделе показано, что многие рецепты, которые мы реализовали по отдельности в главе 3, можно легко выразить в терминах единого обобщенного шаблона. На самом деле, этот шаблон можно применить к вычислению бесконечного разнообразия агрегирующих функций над наборами узлов. К сожалению, такая гибкость и общность даются не бесплатно. Обобщенная реализация как правило работает процентов на 40 медленнее написанного вручную решения. Если быстродействие – важнейший фактор, то стоит прибегнуть к оптимизированному ручному коду. Если же нужно быстро реализовать сложный XSLT-сценарий, в котором выполняется много операций агрегирования, то применение обобщенного подхода может заметно ускорить разработку[1]. Наибольший эффект вы сможете

[1] Не все согласятся с такой оценкой. Кто-то даже возразит, что этот подход лишь замедляет разарботку из-за сложнсти дополтительных уровней косвенности. Однако при многократном использовании сложность начинается казаться идиомой. Вспомните, как вы поначалу сражались с самым обычным кодом на XSLT.

получить, заранее создав широкий набор готовых обобщенных элементов и функций-агрегаторов. В полный вариант таблицы *aggregation.xslt*, который вы найдете на сопроводительном сайте книги (*http://www.oreilly.com/catalog/xsltckbk*), уже включены все функции из этого примера (и ряд других).

В тех случаях, когда агрегирующая функция не симметрична, иногда возникает необходимость выполнять агрегирование набора узлов в обратном порядке. Для этого нужно внести в обобщенную функцию лишь два мелких изменения:

```
<xsl:template name="generic:reverse-aggregation">
  <xsl:param name="nodes"/>
  <xsl:param name="aggr-func" select=" 'sum' "/>
  <xsl:param name="func" select=" 'identity' "/>
  <xsl:param name="func-param1" select="$generic:generics[self::generic:func
                        and @name = $func]/@param1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="accum" select="$generic:generics[self::generic:aggr-func
                    and @name = $aggr-func]/@identity"/>

    <xsl:choose>
      <xsl:when test="$nodes">

       <!-- Вычислить func($x) -->
        <xsl:variable name="f-of-x">
          <xsl:apply-templates select=
          "$generic:generics[self::generic:func and @name = $func]">
            <xsl:with-param name="x" select="$nodes[last( )]"/>
            <xsl:with-param name="i" select="$i"/>
          </xsl:apply-templates>
        </xsl:variable>

       <!-- Агрегировать текущее значение $f-of-x с $accum -->
        <xsl:variable name="temp">
          <xsl:apply-templates
              select="$generic:generics[self::generic:aggr-func and
                          @name = $aggr-func]">
            <xsl:with-param name="x" select="$f-of-x"/>
            <xsl:with-param name="accum" select="$accum"/>
            <xsl:with-param name="i" select="$i"/>
          </xsl:apply-templates>
        </xsl:variable>

        <xsl:call-template name="generic:reverse-aggregation">
          <xsl:with-param name="nodes"
                    select="$nodes[position( )!=last( )]"/>
          <xsl:with-param name="aggr-func" select="$aggr-func"/>
```

```
          <xsl:with-param name="func" select="$func"/>
          <xsl:with-param name="func-param1" select="$func-param1"/>
          <xsl:with-param name="i" select="$i + 1"/>
          <xsl:with-param name="accum" select="$temp"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$accum"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

См. также

В библиотеке FXSL (раздел «См. также» во введении к этой главе) имеются функции `fold` и `foldr`, аналогичные `generic:aggregation` и `generic:reverse-aggregation` соответственно.

16.3. Создание обобщенных функций ограниченного агрегирования

Задача

Требуется написать повторно используемые шаблоны для выполнения широкого спектра операций ограниченного агрегирования.

Решение

```
<xsl:template name="generic:bounded-aggregation">
  <xsl:param name="x" select="0"/>
  <xsl:param name="func" select=" 'identity' "/>
  <xsl:param name="func-param1"/>
  <xsl:param name="test-func" select=" 'less-than' "/>
  <xsl:param name="test-param1" select="$x + 1"/>
  <xsl:param name="incr-func" select=" 'incr' "/>
  <xsl:param name="incr-param1" select="1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="aggr-func" select=" 'sum' "/>
  <xsl:param name="aggr-param1"/>
  <xsl:param name="accum"
             select="$generic:generics[self::generic:aggr-func and
                                       @name = $aggr-func]/@identity"/>

  <!-- Проверяем, нужно ли продолжать агрегирование -->
  <xsl:variable name="continue">
```

```
    <xsl:apply-templates
            select="$generic:generics[self::generic:func and
                                      @name = $test-func]">
      <xsl:with-param name="x" select="$x"/>
      <xsl:with-param name="param1" select="$test-param1"/>
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:choose>
    <xsl:when test="string($continue)">
      <!-- Вычислить func($x) -->
        <xsl:variable name="f-of-x">
          <xsl:apply-templates select="$generic:generics[self::generic:func
                                                         and
                                                         @name = $func]">
        <xsl:with-param name="x" select="$x"/>
        <xsl:with-param name="i" select="$i"/>
        <xsl:with-param name="param1" select="$func-param1"/>
      </xsl:apply-templates>
    </xsl:variable>

    <!-- Агрегировать текущее значение $f-of-x с $accum -->
    <xsl:variable name="temp">
      <xsl:apply-templates
        select="$generic:generics[self::generic:aggr-func and
                                  @name = $aggr-func]">
        <xsl:with-param name="x" select="$f-of-x"/>
        <xsl:with-param name="i" select="$i"/>
        <xsl:with-param name="param1" select="$aggr-param1"/>
        <xsl:with-param name="accum" select="$accum"/>
      </xsl:apply-templates>
     </xsl:variable>

     <!-- Вычислить следующее значение $x-->
     <xsl:variable name="next-x">
    <xsl:apply-templates
         select="$generic:generics[self::generic:func and
                                   @name = $incr-func]">
      <xsl:with-param name="x" select="$x"/>
      <xsl:with-param name="param1" select="$incr-param1"/>
    </xsl:apply-templates>
     </xsl:variable>

     <!-- Рекурсивно обрабатываем остальные узлы, используя position( ) -->
```

```
      <xsl:call-template name="generic:bounded-aggregation">
        <xsl:with-param name="x" select="$next-x"/>
        <xsl:with-param name="func" select="$func"/>
        <xsl:with-param name="func-param1" select="$func-param1"/>
        <xsl:with-param name="test-func" select="$test-func"/>
        <xsl:with-param name="test-param1" select="$test-param1"/>
        <xsl:with-param name="incr-func" select="$incr-func"/>
        <xsl:with-param name="incr-param1" select="$incr-param1"/>
        <xsl:with-param name="i" select="$i + 1"/>
        <xsl:with-param name="aggr-func" select="$aggr-func"/>
        <xsl:with-param name="aggr-param1" select="$aggr-param1"/>
        <xsl:with-param name="accum" select="$temp"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="$accum"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

Этот шаблон выполняет агрегирование не по набору узлов, а по множеству значений. Процедура определяется начальным значением, функцией приращения и предикатом, который говорит, когда следует прекратить агрегирование. Функция приращения и функция проверки могут независимо принимать параметры. Остальные параметры шаблона `generic:bounded-aggregation` такие же, как у шаблона `generic:aggregation` из рецепта 16.2.

Обсуждение

В этом рецепте рассматривается обобщенный подход к задаче агрегирования XML-содержимого. Но иногда возникает необходимость выполнить подобные агрегированию операции над математически синтезированным множеством.

Простейший пример применения обобщенной функции ограниченного агрегирования – это реализация шаблонов `factorial` и `prod-range` из рецепта 3.5:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:aggr="http://www.ora.com/XSLTCookbook/namespaces/aggregate"
  xmlns:exslt="http://exslt.org/common"
  extension-element-prefixes="generic aggr exslt">

  <xsl:import href="aggregation.xslt"/>

  <xsl:output method="xml" indent="yes"/>
```

```
    <xsl:template name="factorial">
      <xsl:param name="n" select="0"/>

      <xsl:call-template name="generic:bounded-aggregation">
        <xsl:with-param name="x" select="$n"/>
        <xsl:with-param name="test-func" select=" 'greater-than' "/>
        <xsl:with-param name="test-param1" select="0"/>
        <xsl:with-param name="incr-func" select=" 'decr' "/>
        <xsl:with-param name="aggr-func" select=" 'product' "/>
      </xsl:call-template>

    </xsl:template>

    <xsl:template name="prod-range">
      <xsl:param name="start" select="1"/>
      <xsl:param name="end" select="1"/>

      <xsl:call-template name="generic:bounded-aggregation">
        <xsl:with-param name="x" select="$start"/>
        <xsl:with-param name="test-func" select=" 'less-than-eq' "/>
        <xsl:with-param name="test-param1" select="$end"/>
        <xsl:with-param name="incr-func" select=" 'incr' "/>
        <xsl:with-param name="aggr-func" select=" 'product' "/>
      </xsl:call-template>

    </xsl:template>

  <xsl:template match="/">

    <results>

      <factorial n="0">
        <xsl:call-template name="factorial"/>
      </factorial>

      <factorial n="1">
        <xsl:call-template name="factorial">
          <xsl:with-param name="n" select="1"/>
        </xsl:call-template>
      </factorial>

      <factorial n="5">
        <xsl:call-template name="factorial">
          <xsl:with-param name="n" select="5"/>
```

```
      </xsl:call-template>
    </factorial>

    <factorial n="20">
      <xsl:call-template name="factorial">
        <xsl:with-param name="n" select="20"/>
      </xsl:call-template>
    </factorial>

    <product start="1" end="20">
      <xsl:call-template name="prod-range">
        <xsl:with-param name="start" select="1"/>
        <xsl:with-param name="end" select="20"/>
      </xsl:call-template>
    </product>

    <product start="10" end="20">
      <xsl:call-template name="prod-range">
        <xsl:with-param name="start" select="10"/>
        <xsl:with-param name="end" select="20"/>
      </xsl:call-template>
    </product>

  </results>

</xsl:template>

</xsl:stylesheet>
```

На выходе получаем такой результат:

```
<results>
    <factorial n="0">1</factorial>
    <factorial n="1">1</factorial>
    <factorial n="5">120</factorial>
    <factorial n="20">2432902008176640000</factorial>
    <product start="1" end="20">2432902008176640000</product>
    <product start="10" end="20">6704425728000</product>
</results>
```

Но это лишь верхушка айсберга! Шаблоном `generic:bounded-aggregation` можно воспользоваться для численного интегрирования:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
```

```
    xmlns:aggr="http://www.ora.com/XSLTCookbook/namespaces/aggregate"
    xmlns:exslt="http://exslt.org/common"
    extension-element-prefixes="generic aggr exslt">

    <xsl:import href="aggregation.xslt"/>

    <xsl:output method="xml" indent="yes"/>

<!-- Расширить набор имеющихся обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

  <xsl:template name="integrate">
    <xsl:param name="from" select="0"/>
    <xsl:param name="to" select="1"/>
    <xsl:param name="func" select=" 'identity' "/>
    <xsl:param name="delta" select="($to - $from) div 100"/>

    <xsl:call-template name="generic:bounded-aggregation">
      <xsl:with-param name="x" select="$from"/>
      <xsl:with-param name="func" select=" 'f-of-x-dx' "/>
      <xsl:with-param name="func-param1">
        <params f-of-x="{$func}" dx="{$delta}"/>
      </xsl:with-param>
      <xsl:with-param name="test-func" select=" 'less-than' "/>
      <xsl:with-param name="test-param1" select="$to"/>
      <xsl:with-param name="incr-func" select=" 'incr' "/>
      <xsl:with-param name="incr-param1" select="$delta"/>
      <xsl:with-param name="aggr-func" select=" 'sum' "/>
    </xsl:call-template>

  </xsl:template>

  <xsl:template name="integrate2">
    <xsl:param name="from" select="0"/>
    <xsl:param name="to" select="1"/>
    <xsl:param name="func" select=" 'identity' "/>
    <xsl:param name="delta" select="($to - $from) div 100"/>

    <xsl:call-template name="generic:bounded-aggregation">
      <xsl:with-param name="x" select="$from"/>
      <xsl:with-param name="func" select=" 'f-of-x-dx-2' "/>
      <xsl:with-param name="func-param1">
        <params f-of-x="{$func}" dx="{$delta}"/>
```

```
    </xsl:with-param>
    <xsl:with-param name="test-func" select=" 'less-than' "/>
    <xsl:with-param name="test-param1" select="$to"/>
    <xsl:with-param name="incr-func" select=" 'incr' "/>
    <xsl:with-param name="incr-param1" select="$delta"/>
    <xsl:with-param name="aggr-func" select=" 'sum' "/>
  </xsl:call-template>

</xsl:template>

<generic:func name="f-of-x-dx"/>
<xsl:template match="generic:func[@name='f-of-x-dx']">
    <xsl:param name="x"/>
    <xsl:param name="param1"/>

    <xsl:variable name="f-of-x">
       <xsl:apply-templates select="$generic:generics[self::generic:func
       and @name = exslt:node-set($param1)/*/@f-of-x]">
          <xsl:with-param name="x" select="$x"/>
       </xsl:apply-templates>
    </xsl:variable>

    <xsl:value-of select="$f-of-x * exslt:node-set($param1)/*/@dx"/>
</xsl:template>

<generic:func name="f-of-x-dx-2"/>
<xsl:template match="generic:func[@name='f-of-x-dx-2']">
    <xsl:param name="x"/>
    <xsl:param name="param1"/>

    <xsl:variable name="func" select="exslt:node-set($param1)/*/@f-of-x"/>
    <xsl:variable name="dx" select="exslt:node-set($param1)/*/@dx"/>

    <xsl:variable name="f-of-x">
       <xsl:apply-templates
          select="$generic:generics[self::generic:func and
                                    @name = $func]">
          <xsl:with-param name="x" select="$x"/>
       </xsl:apply-templates>
    </xsl:variable>

    <xsl:variable name="f-of-x-plus-dx">
       <xsl:apply-templates select="$generic:generics[self::generic:func
                                                      and @name = $func]">
```

```
            <xsl:with-param name="x" select="$x + $dx"/>
          </xsl:apply-templates>
      </xsl:variable>

      <!-- Это просто абсолютное значение $f-of-x-plus-dx - $f-of-x -->
      <xsl:variable name="abs-diff"
            select="(1 - 2 *(($f-of-x-plus-dx - $f-of-x) &lt; 0)) *
                     ($f-of-x-plus-dx - $f-of-x)"/>

      <xsl:value-of select="$f-of-x * $dx + ($abs-diff * $dx) div 2"/>

  </xsl:template>

  <xsl:template match="/">

  <results>

    <integrate desc="intgr x от 0 до 1">
      <xsl:call-template name="integrate"/>
    </integrate>

    <integrate desc="intgr x от 0 до 1 с большей точностью">
      <xsl:call-template name="integrate">
        <xsl:with-param name="delta" select="0.001"/>
      </xsl:call-template>
    </integrate>

    <integrate desc="intgr x от 0 до 1, используя лучший алгоритм">
      <xsl:call-template name="integrate2"/>
    </integrate>

    <integrate desc="intgr x**2 от 0 до 1">
      <xsl:call-template name="integrate">
        <xsl:with-param name="func" select=" 'square' "/>
      </xsl:call-template>
    </integrate>

    <integrate desc="intgr x**2 от 0 до 1, используя лучший алгоритм">
      <xsl:call-template name="integrate2">
        <xsl:with-param name="func" select=" 'square' "/>
      </xsl:call-template>
    </integrate>

  </results>
```

```
</xsl:template>

</xsl:stylesheet>
```

Проблема в том, что пользователь шаблонов интегрирования должен иметь возможность передать любую функцию от x. А нам нужно вычислять сумму, которая является функцией от этой функции. Следовательно, возникает необходимость в функции высшего порядка. К тому же этой функции высшего порядка нужно передавать параметр – в данном случае величину delta, используемую для аппроксимации интеграла. Мы достигаем цели, передавая элемент, синтезированный на лету: `<params f-of-x="{$func}" dx="{$delta}"/>`. Этот составной параметр используется в функциях высшего порядка `f-of-x-dx` и `f-of-x-dx-2`. К сожалению, в XSLT 1.0 для извлечения информации из составного параметра приходится обращаться к функции `exsl:node-set`.

Вот как выглядит результат применения этой таблицы стилей:

```
<results>
  <integrate desc="intgr x от 0 до 1">
    0.4950000000000004
  </integrate>
  <integrate desc="intgr x от 0 до 1 с большей точностью">
    0.4995000000000005
  </integrate>
   <integrate desc="intgr x от 0 до 1, используя лучший алгоритм">
     0.5000000000000001
     </integrate>
   <integrate desc="intgr x**2 от 0 до 1 с большей точностью">
     0.32835000000000036
  </integrate>
  <integrate desc="intgr x**2 от 0 до 1, используя лучший алгоритм">
    0.33335000000000037
  </integrate>
</results>
```

Маловероятно, что вы будете заниматься численным интегрированием с помощью XSLT. Не в этом смысл примера. Я лишь хотел продемонстрировать, чего можно достичь с помощью обобщенного повторно используемого кода.

16.4. Создание обобщенных функций отображения

Задача

Требуется создать повторно используемые шаблоны для выполнения операций над элементами набора узлов.

Решение

В этом решении мы рекурсивно обрабатываем элементы, входящие в набор `$nodes`, вызывая для каждого из них обобщенную функцию `$func`. Эта функция может иметь параметр `$func-param`. Значение `$func-param` по умолчанию мы получаем из атрибута `@param1` метки обобщенной функции. Благодаря этому соглашению значение по умолчанию может зависеть от обобщенной функции:

```
<xsl:template name="generic:map">
  <xsl:param name="nodes" select="/.."/>
  <xsl:param name="func" select=" 'identity' "/>
  <xsl:param name="func-param1"
    select="$generic:generics[self::generic:func and @name = $func]/@param1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="result" select="/.."/>

  <xsl:choose>
    <xsl:when test="$nodes">
      <xsl:variable name="temp">
         <xsl:apply-templates
             select="$generic:generics[self::generic:func and
                                       @name = $func]">
        <xsl:with-param name="x" select="$nodes[1]"/>
        <xsl:with-param name="i" select="$i"/>
        <xsl:with-param name="param1" select="$func-param1"/>
       </xsl:apply-templates>
    </xsl:variable>

    <xsl:call-template name="generic:map">
         <xsl:with-param name="nodes" select="$nodes[position( ) > 1]"/>
         <xsl:with-param name="func" select="$func"/>
         <xsl:with-param name="func-param1" select="$func-param1"/>
         <xsl:with-param name="i" select="$i +1"/>
         <xsl:with-param name="result"
                         select="$result | exslt:node-set($temp)"/>
    </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:apply-templates select="$result" mode="generic:map"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="/ | node( ) | @*" mode="generic:map">
```

```
  <node>
    <xsl:copy-of select="."/>
  </node>
</xsl:template>
```

Как это происходит на практике, можно увидеть на примере обобщенной функции `incr`:

```
    <generic:func name="incr" param1="1"/>
    <xsl:template match="generic:func[@name='incr']">
           <xsl:param name="x"/>
           <xsl:param name="param1" select="@param1"/>
           <xsl:value-of select="$x + $param1"/>
    </xsl:template>
```

Параметр функции `incr` определяет величину приращения и по умолчанию равен 1. Ниже приведена таблица стилей, в которой `incr` применяется к набору узлов, состоящему из чисел. Сначала берется параметр по умолчанию, а потом ему присваивается значение 10. Заодно мы добавляем обобщенную функцию `reciprocal` и применяем ее для отображения множества чисел:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:exslt="http://exslt.org/common"
  extension-element-prefixes="exslt" exclude-result-prefixes="generic">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширяем набор имеющихся обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Добавляем обобщенную функцию для вычисления обратной величины -->
<generic:func name="reciprocal"/>
<xsl:template match="generic:func[@name='reciprocal']">
  <xsl:param name="x"/>
  <xsl:value-of select="1 div $x"/>
</xsl:template>

<!-- Тестируем функциональность отображения -->
<xsl:template match="numbers">

<results>
```

```
<incr>
  <xsl:call-template name="generic:map">
    <xsl:with-param name="nodes" select="number"/>
    <xsl:with-param name="func" select=" 'incr' "/>
  </xsl:call-template>
</incr>
<incr10>
    <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'incr' "/>
      <xsl:with-param name="func-param1" select="10"/>
    </xsl:call-template>
</incr10>
<recip>
    <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'reciprocal' "/>
    </xsl:call-template>
</recip>
</results>

</xsl:template>

</xsl:stylesheet>
```

Ниже приведены результаты работы этой таблицы:

```
    <incr>
            <node>11</node>
            <node>4.5</node>
            <node>5.44</node>
            <node>78.7777</node>
            <node>-7</node>
            <node>2</node>
            <node>445</node>
            <node>2.1234</node>
            <node>8.77</node>
            <node>4.1415927</node>
    </incr>
    <incr10>
            <node>20</node>
            <node>13.5</node>
            <node>14.440000000000001</node>
            <node>87.7777</node>
            <node>2</node>
```

```
            <node>11</node>
            <node>454</node>
            <node>11.1234</node>
            <node>17.77</node>
            <node>13.1415927</node>
    </incr10>
    <recip>
            <node>0.1</node>
            <node>0.2857142857142857</node>
            <node>0.2252252252252252</node>
            <node>0.012857155714298572</node>
            <node>-0.125</node>
            <node>1</node>
            <node>0.0022522522522522522</node>
            <node>0.8901548869503294</node>
            <node>0.1287001287001287</node>
            <node>0.31830988148145367</node>
    </recip>
</results>
```

Обсуждение

Шаблон `map` может извлекать подмножество узлов, удовлетворяющих заданному критерию. Для этого нужно использовать обобщенную функцию-предикат. Требуемый эффект достигается, когда предикат возвращает поданное на вход значение, если условие истинно, и не возвращает ничего, если оно ложно. Например:

```
<generic:func name="less-than"/>
<xsl:template match="generic:func[@name='less-than']">
    <xsl:param name="x"/>
    <!-- верхняя граница -->
    <xsl:param name="param1"/>
    <xsl:if test="$x &lt; $param1"><xsl:value-of select="$x"/></xsl:if>
</xsl:template>
```

Затем полученные узлы запоминаются с помощью следующего фильтрующего шаблона:

```
<xsl:template match="/ | node( ) | @*" mode="generic:map">
  <xsl:if test="string(.)">
    <node>
      <xsl:copy-of select="."/>
    </node>
  </xsl:if>
</xsl:template>
```

Вот как эта техника применяется на практике:

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
   xmlns:exslt="http://exslt.org/common"
   extension-element-prefixes="exslt" exclude-result-prefixes="generic">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Тестируем функциональность отображения -->
<xsl:template match="numbers">

<results>

<less-than-5>
   <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'less-than' "/>
      <xsl:with-param name="func-param1" select="5"/>
   </xsl:call-template>
</less-than-5>

<greater-than-5>
   <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="number"/>
      <xsl:with-param name="func" select=" 'greater-than' "/>
      <xsl:with-param name="func-param1" select="5"/>
   </xsl:call-template>
</greater-than-5>

</results>

</xsl:template>

<xsl:template match="/ | node( ) | @*" mode="generic:map">
   <xsl:if test="string(.)">
      <node>
         <xsl:copy-of select="."/>
      </node>
   </xsl:if>
</xsl:template>
```

Вот результат тестирования этой таблицы стилей:

```
<results>
   <less-than-5>
      <node>3.5</node>
      <node>4.44</node>
      <node>-8</node>
      <node>1</node>
      <node>1.1234</node>
      <node>3.1415927</node>
   </less-than-5>
   <greater-than-5>
      <node>10</node>
      <node>77.7777</node>
      <node>444</node>
      <node>7.77</node>
   </greater-than-5>
</results>
```

Отображение можно применять не только для обработки чисел. Рассмотрим следующую таблицу стилей, которая вычисляет длины всех элементов `para` в документе DocBook:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:exslt="http://exslt.org/common"
  extension-element-prefixes="exslt" exclude-result-prefixes="generic">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширяем набор имеющихся обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Добавляем обобщенную функцию для вычисления длины -->
<generic:func name="length"/>
<xsl:template match="generic:func[@name='length']">
  <xsl:param name="x"/>
  <xsl:value-of select="string-length($x)"/>
</xsl:template>

<!-- Тестируем функциональность отображения -->
<xsl:template match="/">
```

```
<para-lengths>
    <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="//para"/>
      <xsl:with-param name="func" select=" 'length' "/>
    </xsl:call-template>
</para-lengths>

</xsl:template>

<xsl:template match="/ | node( ) | @*" mode="generic:map">
  <length>
    <xsl:copy-of select="."/>
  </length>
</xsl:template>

</xsl:stylesheet>
```

Или такой пример, где создается реферат документа путем извлечения первых трех предложений из каждого абзаца:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic"
  xmlns:exslt="http://exslt.org/common"
  extension-element-prefixes="exslt" exclude-result-prefixes="generic">

<xsl:import href="aggregation.xslt"/>

<xsl:output method="xml" indent="yes"/>

<!-- Расширяем набор имеющихся обобщенных функций -->
<xsl:variable name="generic:generics" select="$generic:public-generics |
document('')/*/generic:*"/>

<!-- Обобщенная функция для извлечения предложений -->
<generic:func name="extract-sentences" param1="1"/>
<xsl:template match="generic:func[@name='extract-
sentences']" name="generic:extract-sentences">
  <xsl:param name="x"/>
  <xsl:param name="param1" select="@param1"/>
   <xsl:choose>
    <xsl:when test="$param1 >= 1 and contains($x,'.')">
      <xsl:value-of select="substring-before($x,'.')"/>
      <xsl:text>.</xsl:text>
      <xsl:call-template name="generic:extract-sentences">
        <xsl:with-param name="x" select="substring-after($x,'.')"/>
```

```
        <xsl:with-param name="param1" select="$param1 - 1"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
</xsl:template>

<xsl:template match="/">

<summary>
    <xsl:call-template name="generic:map">
      <xsl:with-param name="nodes" select="//para"/>
      <xsl:with-param name="func" select=" 'extract-sentences' "/>
      <xsl:with-param name="func-param1" select="3"/>
    </xsl:call-template>
</summary>

</xsl:template>

<xsl:template match="/ | node( ) | @*" mode="generic:map">
  <para>
    <xsl:copy-of select="."/>
  </para>
</xsl:template>

</xsl:stylesheet>
```

Эти примеры – чрезмерно усложненный способ получения результата, который можно было бы получить гораздо проще. Создать таблицу стилей, которая применяет преобразования последовательно к каждому узлу, легко. Например, таблицу для формирования реферата можно реализовать и так:

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

<xsl:template name="extract-sentences">
  <xsl:param name="text"/>
  <xsl:param name="num-sentences" select="1"/>
   <xsl:choose>
    <xsl:when test="$num-sentences >= 1 and contains($text,'.')">
      <xsl:value-of select="substring-before($text,'.')"/>
      <xsl:text>.</xsl:text>
      <xsl:call-template name="extract-sentences">
        <xsl:with-param name="text" select="substring-after($text,'.')"/>
        <xsl:with-param name="num-sentences" select="$num-sentences - 1"/>
```

```
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise/>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="/">
    <summary>
      <xsl:apply-templates select=".//para"/>
    </summary>
  </xsl:template>

  <xsl:template match="para">
    <para>
        <xsl:call-template name="extract-sentences">
          <xsl:with-param name="text" select="."/>
          <xsl:with-param name="num-sentences" select="3"/>
        </xsl:call-template>
      </para>
  </xsl:template>

  </xsl:stylesheet>
```

Однако, если в одной таблице стилей нужно выполнить несколько отображений, то применение обобщенной реализации позволит уменьшить объем специализированного кода.

Рассмотрим еще обобщенную функцию отображения `generic:map2`. Вместо того чтобы применять унарную функцию к одному набору узлов, `generic:map2` применяет бинарную функцию к узлам из двух параллельных наборов и возвращает результирующий набор:

```
<xsl:template name="generic:map2">
  <xsl:param name="nodes1" select="/.."/>
  <xsl:param name="nodes2" select="/.."/>
  <xsl:param name="func" select=" 'identity' "/>
  <xsl:param name="func-param1" select="$generic:generics[self::generic:func
and @name = $func]/@param1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="result" select="/.."/>

  <xsl:choose>
    <xsl:when test="$nodes1 and $nodes2">
      <xsl:variable name="temp">
        <xsl:apply-templates
             select="$generic:generics[self::generic:aggr-func and
```

```
                                        @name = $func]">
            <xsl:with-param name="x" select="$nodes1[1]"/>
            <xsl:with-param name="accum" select="$nodes2[1]"/>
            <xsl:with-param name="i" select="$i"/>
            <xsl:with-param name="param1" select="$func-param1"/>
          </xsl:apply-templates>
        </xsl:variable>

       <xsl:call-template name="generic:map2">
        <xsl:with-param name="nodes1" select="$nodes1[position( ) > 1]"/>
        <xsl:with-param name="nodes2" select="$nodes2[position( ) > 1]"/>
        <xsl:with-param name="func" select="$func"/>
        <xsl:with-param name="func-param1" select="$func-param1"/>
        <xsl:with-param name="i" select="$i +1"/>
        <xsl:with-param name="result" select="$result | exslt:node-
set($temp)"/>
       </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:apply-templates select="$result" mode="generic:map"/>
      </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

Как и в случае `generic:map`, полезность функции `generic:map2` зависит от того, как часто она применяется.

16.5. Создание обобщенных генераторов наборов узлов

Задача

Требуется написать повторно используемые шаблоны для динамической генерации набора узлов.

Решение

Первая обобщенная функция из этой категории генерирует набор узлов путем вызова для значений, последовательность которых определяется функцией приращения. Процедура повторяется, пока не будет достигнута верхняя граница:

```
<xsl:template name="generic:gen-set">
  <xsl:param name="x" select="1"/>
  <xsl:param name="func" select=" 'identity' "/>
```

```
  <xsl:param name="func-param1"
    select="$generic:generics[self::generic:func and @name = $func]/
@param1"/>
  <xsl:param name="test-func" select=" 'less-than' "/>
  <xsl:param name="test-param1" select="$x + 1"/>
  <xsl:param name="incr-func" select=" 'incr' "/>
  <xsl:param name="incr-param1" select="1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="result" select="/.."/>

  <!-- Проверяем, нужно ли продолжать генерацию -->
  <xsl:variable name="continue">
    <xsl:apply-templates
        select="$generic:generics[self::generic:func and
                @name = $test-func]">
      <xsl:with-param name="x" select="$x"/>
      <xsl:with-param name="param1" select="$test-param1"/>
    </xsl:apply-templates>
  </xsl:variable>

  <xsl:choose>
    <xsl:when test="string($continue)">
      <!-- Вычислить func($x) -->
       <xsl:variable name="f-of-x">
         <xsl:apply-templates
              select="$generic:generics[self::generic:func and
                                        @name = $func]">
        <xsl:with-param name="x" select="$x"/>
        <xsl:with-param name="i" select="$i"/>
        <xsl:with-param name="param1" select="$func-param1"/>
       </xsl:apply-templates>
      </xsl:variable>

      <!-- Вычислить следующее значение $x-->
      <xsl:variable name="next-x">
        <xsl:apply-templates
              select="$generic:generics[self::generic:func and

        <xsl:with-param name="x" select="$x"/>
        <xsl:with-param name="param1" select="$incr-param1"/>
       </xsl:apply-templates>
      </xsl:variable>

      <xsl:call-template name="generic:gen-set">
       <xsl:with-param name="x" select="$next-x"/>
```

```
        <xsl:with-param name="func" select="$func"/>
        <xsl:with-param name="func-param1" select="$func-param1"/>
        <xsl:with-param name="test-func" select="$test-func"/>
        <xsl:with-param name="test-param1" select="$test-param1"/>
        <xsl:with-param name="incr-func" select="$incr-func"/>
        <xsl:with-param name="incr-param1" select="$incr-param1"/>
        <xsl:with-param name="i" select="$i + 1"/>
        <xsl:with-param name="result"
                        select="$result | exslt:node-set($f-of-x)"/>
       </xsl:call-template>
     </xsl:when>
     <xsl:otherwise>
        <xsl:apply-templates select="$result" mode="generic:gen-set"/>
     </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="node( )" mode="generic:gen-set">
  <gen-set>
    <xsl:copy-of select="."/>
  </gen-set>
</xsl:template>
```

Ниже этот шаблон используется для генерации последовательности квадратов первых десяти целых чисел:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic">

  <xsl:import href="aggregation.xslt"/>

  <xsl:output method="text" />

  <xsl:template match="/">
    <xsl:call-template name="generic:gen-set">
      <xsl:with-param name="x" select="1"/>
      <xsl:with-param name="func" select=" 'square' "/>
      <xsl:with-param name="incr-param1" select="1"/>
      <xsl:with-param name="test-func" select=" 'less-than-eq' "/>
      <xsl:with-param name="test-param1" select="10"/>
    </xsl:call-template>
  </xsl:template>

<xsl:template match="node( )" mode="generic:gen-set">
<xsl:value-of select="."/>
```

```
<xsl:text> </xsl:text>
</xsl:template>

1 4 9 16 25 36 49 64 81 100
```

Вторая обобщенная функция, которую мы рассмотрим, генерирует набор узлов, n раз вызывая некоторую функцию, причем при первом вызове ей передается заданное начальное значение, а при каждом последующем – результат предыдущего вызова:

```
<xsl:template name="generic:gen-nested">
  <xsl:param name="x" select="1"/>
  <xsl:param name="func" select=" 'identity' "/>
  <xsl:param name="func-param1"
             select="$generic:generics[self::generic:func and
                     @name = $func]/@param1"/>
  <xsl:param name="i" select="1"/>
  <xsl:param name="n" select="2"/>
  <xsl:param name="result">
    <xsl:value-of select="$x"/>
  </xsl:param>

    <xsl:choose>
      <xsl:when test="$i &lt;= $n">
       <!-- Вычислить func($x) -->
        <xsl:variable name="f-of-x">
          <xsl:apply-templates
              select="$generic:generics[self::generic:func and
                                        @name = $func]">
            <xsl:with-param name="x" select="$x"/>
            <xsl:with-param name="i" select="$i"/>
             <xsl:with-param name="param1" select="$func-param1"/>
          </xsl:apply-templates>
        </xsl:variable>

          <xsl:call-template name="generic:gen-nested">
            <xsl:with-param name="x" select="$f-of-x"/>
            <xsl:with-param name="func" select="$func"/>
            <xsl:with-param name="func-param1" select="$func-param1"/>
            <xsl:with-param name="i" select="$i + 1"/>
            <xsl:with-param name="n" select="$n"/>
            <xsl:with-param name="result"
                  select="exslt:node-set($result) |
                  exslt:node-set($f-of-x)"/>
          </xsl:call-template>
```

```
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="$result" mode="generic:gen-nested"/>
        </xsl:otherwise>
      </xsl:choose>
</xsl:template>

<xsl:template match="node( )" mode="generic:gen-nested">
  <gen-nested>
    <xsl:copy-of select="."/>
  </gen-nested>
</xsl:template>
```

Ниже этот шаблон используется для построения последовательности 2, 2 ** 2, (2 ** 2) ** 2, ((2 ** 2) ** 2) ** 2, (((2 ** 2) ** 2) ** 2) ** 2, где ** означает возведение в степень:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic">

  <xsl:import href="aggregation.xslt"/>

  <xsl:output method="text" />

  <xsl:template match="/">
    <xsl:call-template name="generic:gen-nested">
      <xsl:with-param name="x" select="2"/>
      <xsl:with-param name="func" select=" 'square' "/>
      <xsl:with-param name="n" select="4"/>
    </xsl:call-template>
  </xsl:template>

<xsl:template match="node( )" mode="generic:gen-nested">
  <xsl:value-of select="."/>
  <xsl:text> </xsl:text>
</xsl:template>

</xsl:stylesheet>

2 4 16 256 65536
```

Обсуждение

В рецептах 16.2 и 16.3 приведены преобразования вида многие-к-одному, а в рецепте 16.5 речь шла о преобразованиях вида многие-к-многим. Естественно, эту главу нельзя было бы считать полной без рассмотрения обобщенного преобразования один-к-одному.

С помощью генератора можно порождать случайные числа для отбора случайных узлов из XML-документа. В этой главе описывается простой *линейно-конгруэнтный генератор* (см., например, *http://www.taygeta.com/rwalks/node1.html*). Следующая таблица стилей выводит случайную выборку имен из входного документа:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:generic="http://www.ora.com/XSLTCookbook/namespaces/generic">

  <xsl:import href="aggregation.xslt"/>

  <xsl:output method="xml" indent="yes"/>

  <!-- Расширить набор имеющихся обобщенных функций -->
  <xsl:variable name="generic:generics" select="$generic:public-generics
  | document('')/*/generic:*"/>

  <!-- При таких начальных значениях получаются достаточно случайные -->
  <!-- результаты, но можете поэкспериментировать с другими -->
  <xsl:variable name="a" select="16807"/>
  <xsl:variable name="c" select="0"/>
  <xsl:variable name="m"  select="2147483647"/>

   <!-- Сохранить корневой элемент для последующего использования -->
   <xsl:variable name="doc" select="/"/>

   <!-- Генератор случайных чисел -->
   <generic:func name="linear-congruential"/>
   <xsl:template match="generic:func[@name='linear-congruential']">
     <xsl:param name="x"/>
     <xsl:value-of select="($a * $x + $c) mod $m"/>
   </xsl:template>

   <xsl:template match="/">
   <names>
     <xsl:call-template name="generic:gen-nested">
       <xsl:with-param name="x" select="1"/>
       <xsl:with-param name="func" select=" 'linear-congruential' "/>
       <xsl:with-param name="n" select="100"/>
       <!-- Не включаем начальное значение -->
       <xsl:with-param name="result" select="/.."/>
     </xsl:call-template>
   </names>
</xsl:template>

<xsl:template match="node( )" mode="generic:gen-nested">
```

```
    <!-- Ограничиваемся диапазоном от 1 до 100 -->
    <xsl:variable name="random" select=". mod 99 + 1"/>

      <name>
      <xsl:value-of select="$doc/names/name[$random]"/>
    </name>

</xsl:template>

</xsl:stylesheet>
```

Алфавитный указатель

А

Абсолютное значение (математические функции) 100
Абсолютный номер дня, вычисление 151
Автоматизация
  вставки отладочной печати 780
  генерации кода 552
    CGI-сценарий для тестирования данных 592
    XSLT из XSLT 617
    заглушка обработчика сообщения 573
    из UML-моделей 598
    клиент для тестирования данных 590
    обертки для данных 576
    отладка 582
    предложение switch 567
    файл констант 562
Агрегирование
  ограниченное 830
  последовательностей 221
  элементов 816
Адресация, XPath 17
Алгоритмы
  обобщенные 800
  построения визуального представления дерева 535
  разделяй и властвуй 52
  сортировки 676
  тематическая карта 676
Арабские цифры
форматирование 89
Ассоциации
  UML 654
  отношения 668, 672
Атомарные значения
преобразование 237
Атрибуты
  @name 695
  disable-output-escaping 756
  extension-element-prefixes 747
  format 250
  group-adjacent 231
  group-by 231
  group-starting-with 232
  href 250
  type 250
  validation 250
  xmi.id 656
  xmi.idref 656
  lang 181
  календари 181
  переименование 330
  стилизация с помощью 459

Б

Безье, кривые 506
Битовые маски 134

В

Вывод
  нескольких документов 249
  нескольких документов в Saxon 716
Выражения
  if (XPath 2.0) 30
  XPath 731
  в атрибуте group-by 231
  путевые с использованием осей 23
  регулярные 47
    обработка неструктурированного текста 244
    сопоставление с текстовыми образцами 74
    эмуляция 73
  фильтры 26
Вычисление
  комбинаторных функций 132
  статистических функций 128
  сумм и произведений 114
  суммы полиморфных функций 826

Г

Гиперссылки
  абсолютные 444
  в HTML 441
Граничные условия, тестирование 796
Группировка
  данных для представления в табличном виде 446
  оптимизация 230
  по значениям 231

по конечному узлу 232
по начальному узлу 232
по соседним узлам 231

Д

Двоичные деревья 210
Действия 600
Денежные суммы, форматирование 85
Диаграммы
генерация в формате SVG 487
секторные 512
столбчатые 499
Динамические ошибки 180, 184, 731
Документы
Visio 633
XML
заполнение HTML-форм 471
контейнер 343
объединение документов с одинаковой схемой 337
объединение документов с различными схемами 343
расщепление 349
реорганизация иерархии 361
соединение 391
соединение элементов 348
создание HTML-документов, связанных гиперссылками 441
создание HTML-таблиц 444
создание HTML-фреймов 453
создание замкнутого преобразования в HTML 466
сплетение 345
углубление иерархии 355
уплощение иерархии 352
вывод нескольких 249
порядок 37
удаление пространств имен 337
Дуги рисование 512

З

Заглушки
генерация кода 573
Запросы
к XML-документу 367
из спецификации W3C XML 397
определение равенства множеств по значению 384
с семантикой значения 372
соединения 391
сохраняющие структуру 389
основанные на ссылках 422
рекурсивные 420

И

Иерархии
отображение 297
представление в SVG 529
реорганизация 361
углубление 355
уплощение 352
Индикаторы предметов 672
Интегрирование
численное 834
шаблон 838

К

Календари
ISO 158
атрибуты 181
вычисление последнего дня месяца 144
григорианский 144
еврейский 163
исламский 161
Хиджры 163
юлианский 157
Классы
AnsweringMachine State 600
CkBkTemplParam 753
Graphics2D 742
java.awt.Graphics 744
TextLayout 742
UML 653
XslForEach 754
темы 657
обертки данных, генерация 576
символов 47
Код
встроенный сценарий 713
обобщенный 800
тестовый 792
устранение избыточности 234
Колонки
Excel 652
создание отчетов 285
форматирование для вывода в несколько 83
Комментарии, отладочные 773
Компоненты (XML) 69
Конвейер преобразований 718
Конечный автомат 598
символы экранирования 753

Копирование
пространств имен 336
Корневая тема 681

Л

Логарифмы, вычисление 102

М

Маклорена, ряд 110
Межпроцессные сообщения
протоколы 553
репозиторий 554
Метасимволы в регулярных выражениях 47
Метонов цикл 164
Модификатор представления 183
Мюнха
метод 233, 363, 450

О

Обертки данных 576
Обобщения отношение 669
Обобщенное программирование 805
Обобщенные генераторы наборов узлов 848
Обобщенные функции
агрегирования элементов 816
ограниченного агрегирования 830
отображения 838
Обратная совместимость 238
Обход дерева 191
в обратном порядке 207
в прямом порядке 202
во внутреннем порядке 210
обработка узлов
по позиции 220
по уровням 214
Округление чисел 90
Операторы
| 36
except 37
XPath 17
логические 26
сравнения 24
Оси
x и y 488
братьев 20
дочерняя 18
последователей 23
потомков 18
предков 22
предшественников 23
применение 18
родительская 22
Отладка
message, элемент 770
комментарии 773
преобразование таблицы стилей с добавлением отладочной печати 780
приложений 581
утверждения 772
Отношения
ассоциации 672
обобщения 669
Ошибки
динамические 180, 184, 731
исправимые 184
проверка входных данных 796
статические 180
типизации 24

П

Параметры
pageNumber 627
userScale 627
туннельные 240
удаление лишних 238
Паттерны
Template Method 307
конечные автоматы 598
объектно-ориентированные, эмуляция 239
Перегрузка 808
Переопределение 808
идиома переопределяемого копирования 322, 331
шаблонов 499, 509, 801
Пиза метод 56
Полиморфизм 807
вычисление сумм 826
Полное представление значения 185
Порты 700
Последовательности
агрегирование 32
отображение 32
порождение 33
преобразование в атомарные значения 237
проверка условия 34
расширение 33
соединение 33
узлов 27
обработка по позиции 220
Праздники
определение даты 187
Преобразование
SVG в XML 543
XMI в XTM 653
XML в текст 259
абсолютных дней 158
атомарных значений 237
атрибутов в элементы 323
булевских значений в числовые 102
простых именованных шаблонов 228

расщепление 718
регистра 67
режимы 442
римских числительных 93
системы счисления 96
текста в XML 76
типов 238
тождественное 322
формата Visio VDX в SVG 626
элементов в атрибуты 326
Привязка
портов 700
пространства имен 710
Приложения
Java 765
отладка 582, 724
репозиторий 554
Пространства имен
изменение имени 332
копирование 336
при объявлении
Java-расширений 711
Протоколы
SOAP 626
обмена сообщениями 553
Пустое пространство
вставка символов
новой строки 263
добавление 262
удаление 45, 260
формальные правила 263
Пустой набор узлов 27

Р

Разбиение на части 115
Разбор
текста 76
функции 590
Разрыв строки,
форматирование 263
XML
выполнение теоретико-
множественных
операций над
наборами 368
Запросы
к XML-документу
выполнение теоретико-
множественных
операций над
наборами 367, 368
Расширение текста 75
Расширения 709
EXSLT 76
в Saxon 709
использование 716
элементы 710
в Xalan 716
элементы 713
на языке Java
в формате Java 712
в формате класса 711
в формате пакета 712
реализация 738
функции 738
элементы 713, 746
на языке JavaScript 732
написание 729
содержимого глобальных
переменных 801
функции в MSXML 715
элементы, на сценарном
языке 714
Регулярные выражения
обработка
неструктурированного
текста 244
применение 47
сопоставление
с текстовым образцом 74
эмуляция 73
Режимы (атрибут
mode) 234, 442
Рекурсивное
агрегирование 221, 224
Рекурсия
разбиение на части 115
разделяй и властвуй 115
рекурсивные запросы 420
хвостовая 59, 61
Реляционные базы
данных 725

С

Сворачивание ветвей
дерева 305
Сериализация документа 248
Сетка, создание 495
Сжатие файлов 746
Символы
арабские,
форматирование 89
классы 47
неразрываемый
пробел 263
новой строки,
вставка 263
подсчет числа вхождений
в строку 73
таблицы, применение для
сериализации 248
удаление 45
Система счисления,
преобразования 96
Служебные параметры 105
События стиля 780
Соединение
последовательностей 33
элементов 348
Соединение по
равенству 394
Сопоставление
в языке XPath 17
с шаблоном 804
эмуляция регулярных
выражений 73
эмуляция с текстовым
образцом 74
Сочетания (комбинаторные
функции) 132
Спецификатор
компонента 183

Стереотипы 599, 653
variant 654
идентификация 656
ключи 656
Столбчатые диаграммы горизонтальные 504
Стражи 600
Строки 43
URI-кодирование 78
выравнивание 77
замены подстроки 77
заполнение 77
изменение регистра символов 67
литералы, содержащие кавычки 251
обращение 57
повторение 53
подсчет числа вхождений 73
поиск
методом «разделяй и властвуй» 52, 72
разделителей слов 62
разбиение 78
разбиение на лексемы 70
расширения EXSLT 76
удаление пробелов 46
форматные 183

Т

Таблицы стилей
внедрение 469
для DocBook 815
для представления структуры XML-документа в виде дерева 535
для преобразования из формата Excel XML 644
для создания реферата документа 845
секторных диаграмм 518
команда обработки xml-stylesheet 433, 469
копирование 322
отладка 769, 780
переносимость 237
создание 618
тождественное преобразование 322
трассировка 773
управление пустым пространством 263
Текст 244
анализ 76
вывод 259
с нумерацией 307
замена 61, 75
перенос 317
пустое пространство 259
расширение 75
с отступами 297
сопоставление с образцом 73
форматирование в несколько колонок 285
Темы
естественное имя 671
классы 657
соглашение об отображении в UML 653, 654
Теоретико-множественные операции
EXSLT 371
применение 36
реализация 723
Тождественное преобразование 322
Точка вклинивания, вызов именованного шаблона 809

У

Углубление иерархии 355
Удаление
дубликатов 338
лишних параметров 238
пробелов 46, 259
символов из строки 45
Узлы
выполнение теоретико-множественных операций 368
группировка 231
запоминание 842
обобщенные генераторы 848
обработка по позиции 220
отбор по контексту 200
отбор уникальных 194
поиск минимума и максимума 119
получение расширенной информации 724
последовательности 27
сравнение 36
фильтрация 24
Уплощение иерархии 352
Условные директивы include/import 623
Утверждения 772

Ф

Факториал (математическая функция) 110
Фильтрация узлов 24
Форматирование
в виде HTML-таблиц 444
в несколько колонок 285
времени 173
дат 173
денежных сумм 85
запись числа арабскими цифрами 89
запись числа римскими цифрами 87, 93
пустое пространство 259
разрыв строки 263
текста
перенос 317
формирование отступов 297

чисел 81
экспорт в CSV-файл 266
Форматная строка 183
Функции
abs 111
ckbk:log() 102
ckbk:log10() 102
ckbk:lowest() 123
ckbk:min() 121
ckbk:power() 106
ckbk:power-f() 108
concat 55
contains() 62
current() 256
current-group() 230
current-grouping-key() 230
document() 257
ends-with() 44
floor() 144
format-date() 148, 181
format-number() 81, 92
function-available() 257
incr(), обобщенная 840
key() 257
last() 220
lower-case() 67
name() 200
node-set()() 196
normalize-space() 46
position() 220
replace() 47
saxon:path() 731
starts-with() 44
str:align() 77
str:concat() 78
str:decode-uri() 78
str:encode-uri() 78
str:padding() 77
str:replace() 77
str:split() 78
str:tokenize() 76
string-length() 44
substring() 44
substring-after() 64
substring-before() 52, 64
sum() 118
system-property() 257
tokenize() 50, 72
translate() 45, 67, 73
unparsed-text() 246
upper-case() 67
XPath 17
безопасные
относительно типа 235
для работы со
временем 141
математические 100
вычисление сумм
и произведений 114
комбинаторные 132
минимум
и максимум 119
проверка битов 134
статистические 128
отображения,
обобщенные 838
полиморфизм 807
получения расширенной
информации об узле 724
разбора 590
расширения 76
на сценарном
языке 713
написание 729
расширения в MSXML 715
расширения в Saxon 709
расширения в Xalan-
Java 2 711
Функциональное
программирование 805

Ц

Цвет, настройка 499
Целочисленная арифметика,
эмуляция 144

Ш

Шаблоны
algorithm-page 689
BarStyle 501
bitTest 134
ckbk
log10-util 105
power-frac 110
difference 376
intersection 374
SVG 481
union 374
visio-nurbs.xsl 638
граничные условия 796
для численного
интегрирования 838
метки 803
низкоприоритетные,
добавление поведения 244
переопределение 499,
509, 801
преобразования
юлианского дня 152
простые именованные,
преобразование
в функции 228
рисования осей
координат 494
форматирования 683

Э

Экранирование 751
атрибут disable-output-
escaping 756
конечный автомат 753
Экспорт
Excel в XML 641
XML в файл
с разделителями
полей 265
Электронные таблицы
имитация 88
экспорт в формате XML 641

Элементы
cell 274
ckbk
abs(x) 100
sqrt(x) 101
Geom 638
saxon:output 350
ss:Worksheet 651
svgElement 633
xinclude 457
xsl:apply-templates 252
xsl:attribute 253
xsl:call-template 253
xsl:comment 253
xsl:copy 253, 336
xsl:copy-of 253, 336, 343
xsl:date-format 180
xsl:decimal-format 81
xsl:element 254, 331
xsl:for-each 254
xsl:for-each-group 230
xsl:include 792
xsl:key 339, 393
xsl:message 254, 770
xsl:number 82, 95, 255, 315
xsl:output 255
xsl:param 255
xsl:preserve-space 262
xsl:processing-instruction 255
xsl:strip-space 256, 260
xsl:stylesheet 256
xsl:template 256
xsl:text 261, 262
xsl:value-of 256
xsl:variable 256
xsl:with-param 256
xslx:templtext 753
в Xalan Java 2 713
и вертска страницы 683
игнорирование дубликатов 194
обобщенные функции агрегирования 816
переименование 330
преобразование в атрибуты 326
расширения
реализация на Java 746
реализация на сценарном языке 714
расширения в Saxon 710
соединение 348, 391

Я

Ячейки таблицы 274

A

abs(), функция 111
Adobe SVG, надстройка 551
AnsweringMachineState, класс 600
Antenna House XSL Formatter 432
Apache Cocoon 431

B

baseName, стереотип 653

C

catchXSL! 780
Chain of responsibility, паттерн 243
CkBkTemplParam, базовый класс 753
Cocoon 692
Cogitative Topic Maps Web Site (CTW) 676
CSV, формат 266

D

Decorator, паттерн 244
DocBook, схема 437, 815
DTD (определение типа документа) 419

E

Excel, экспорт в XML 641
expanded-QName 181
EXSLT 44
операции со строками 76
теоретико-множественные операции над узлами 371

I

IDEAlliance 671
IEEE, ограничения 92
instanceOf, отношение (XTM) 663

M

MathML, язык 206
MSXML, функции расширения 715

N

NIST (Национальный институт стандартов и технологий США) 678
NURBS (неоднородные рациональные B-сплайны) 638

P

Perl, и XSLT 763
PSI (публикуемые индикаторы предметов) 669

Q

QName 180

R

Rational Rose 654

S

SAX (Simple API for XML) 587
Saxon
  вывод в несколько файлов 716
  методы класса StyleElement 747, 748
  расширения 709
  трассировка 773
SGML (Standard Generalized Markup Language) 418
SOAP (Simple Object Access Protocol) 626
SVG
  встраивание в HTML-код 541
  графическое представление деревьев 529
  повторно используемые утилиты генерации 487
  преобразование из формата Visio VDX 626
  преобразование имеющейся заготовки 479

T

TrAX (Transformation API for XML) 766

U

UML (унифицированный язык моделирования) 561
  ассоциации 668
  генерация кода 598
  генерация тематических карт в формате XTM 653
  классы 653

V

Visio
  главные шаблоны 633
  диаграммы 640
  преобразование VDX в SVG 626

W

WSDL (Web Service Definition Language) 626
  генерация документации 692

X

Xalan-Java 2, расширения 711, 716
XInclude 345
XMI (XML Metadata Interchange) 561
XML (Extensible Markup Language)
  Excel, экспорт 641
  SAX 587
  атрибуты
    переименование 330
    преобразование в элементы 323
  документы
    заполнение HTML-форм 471
    расщепление 349
    реорганизация иерархии 361
    с одной и той же схемой, объединение 337
    с различными схемами, объединение 343
    создание HTML-документов, связанных ссылками 441
    создание HTML-таблиц 444
    создание HTML-фреймов 453
    создание замкнутого преобразования в HTML 466
    таблицы стилей, управляемые данными 459
    углубление иерархии 355
    уплощение иерархии 352
  запросы 367
    из спецификации W3C XML 397
    семантика значений 372
    соединение документов 391
    сохраняющие структуру 389
    сравнение наборов узлов на равенство 384
  узлы
    отбор 18
    последовательности 27
    фильтрация 24
  экспорт в файл с разделителями полей 265
  элементы
    переименование 330
    преобразование в атрибуты 326
xml-stylesheet, команда обработки 433
XPath
  выражения if 29
  динамическое вычисление выражений 623
  оси 18
  типы 38
  узлы
    обзор 17
    операции над множествами 36

последовательности 27
сравнение 37
фильтрация 24
функция sum() 118
XQuery 367
XSLT
генерация таблиц
стилей 617
новые возможности 252
XSmiles 432

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(495) 258-91-94, 258-91-95**; электронный адрес **books@alians-kniga.ru**.

Сэл Мангано

XSLT. Сборник рецептов

Главный редактор *Мовчан Д. А.*
dm@dmk-press.ru
Перевод с английского *Слинкин А. А.*
Верстка *Старцевой Е. М.*
Корректор *Синяева Г. И.*
Дизайн обложки *Мовчан А. Г.*

Совместный проект издательства «ДМК Пресс» и издательства «БХВ-Петербург»

Подписано в печать 06.04.2008. Формат 70×100 $^{1}/_{16}$.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 31,5. Тираж 1500 экз.
№

Издательство ДМК Пресс
Электронный адрес издательства: www.dmk-press.ru
Internet-магазин: www.alians-kniga.ru

Санитарно-эпидемиологическое заключение на продукцию
№77.99.60.953.Д.002108.02.07
от 28.02.2007 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12