Решения для разработчиков приложений
на платформе Adobe® Flash® и Adobe Flex™

ActionScript 3.0
Сборник рецептов

Символ®
O'REILLY®

Джои Лотт,
Деррон Шалл,
Кейт Питерс

ActionScript 3.0 Cookbook™

Joey Lott, Darron Schall, and Keith Piters

O'REILLY®

ActionScript 3.0
Сборник рецептов

Джои Лотт, Деррон Шалл
и Кейт Питерс

Санкт-Петербург — Москва
2008

Джои Лотт, Деррон Шалл и Кейт Питерс

ActionScript 3.0. Сборник рецептов

Перевод Н. Шатохиной

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Антипин*
Редактор	*В. Овчинников*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Лотт Дж., Шалл Д., Питерс К.
ActionScript 3.0. Сборник рецептов. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 608 с., ил.

ISBN-10: 5-93286-091-X
ISBN-13: 978-5-93286-091-5

Если вам надо быстро найти решение при работе с программным обеспечением Adobe Flash ® или Adobe Flex™, возьмите «ActionScript 3.0. Сборник рецептов». Более 300 рецептов этого практического инструментария дадут ответы на все вопросы, касающиеся ActionScript 3.0. Среди них: определение версии Flash Player или операционной системы пользователя, форматирование дат и представление сумм в разных валютах, обработка пользовательского ввода и работа с текстовыми строками, создание графических объектов во время выполнения, работа с аудио- и видеоданными, удаленный вызов процедур с использованием технологии Flash Remoting, а также загрузка, отправка и поиск XML-данных.

Книга адресована разработчикам во Flash и Flex 2 и содержит готовые решения задач, с которыми они ежедневно сталкиваются. Все рецепты снабжены кратким комментарием, поясняющим механизм работы рекомендуемого программного кода, что облегчает его адаптирование под конкретную задачу.

ISBN-10: 5-93286-091-X
ISBN-13: 978-5-93286-091-5
ISBN-10: 0-596-52695-4 (англ)
ISBN-13: 978-0-596-52695-5 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 30.10.2007. Формат 70x100 1/16 . Печать офсетная.
Объем 38 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Книга «ActionScript 3.0. Сборник рецептов» адресована разработчикам на ActionScript любого уровня. Она написана для обычных людей, которым нужны практические решения часто встречающихся задач. Положите эту книгу рядом с компьютером, и она поможет ответить на вопросы, которые возникают в ходе разработки программных продуктов. Она полна реальных примеров, замечательных и доступных решений, предлагает анализ ситуаций, с которыми, несомненно, сталкиваются разработчики во Flash и на ActionScript.

Книга написана в классическом для серии «O'Reilly Cookbook» (Сборники рецептов O'Reilly) формате. Каждый рецепт представлен разделами «Задача», «Решение» и «Обсуждение». Читатель может быстро найти рецепт, наиболее соответствующий его ситуации, и получить ответ. Чтобы понять предлагаемый код, ему не придется перечитывать всю книгу. В разделе «Обсуждение» каждого рецепта приводится углубленный анализ решения, обсуждаются его возможные варианты и последствия их применения. Таким образом, вы убиваете сразу двух зайцев – получаете быстрый и легкий доступ к необходимым ответам и возможность глубже взглянуть на природу как проблемы, так и решения. «ActionScript 3.0. Сборник рецептов» помогает понять основные принципы на реальных примерах.

Чего нет в этой книге

В данной книге представлен большой объем информации по широкому диапазону тем. Она охватывает весь спектр вопросов по применению ActionScript для создания программных продуктов, работающих на клиентской стороне. Каждый рецепт приводится в контексте практического примера. Необходимая теория, конечно, дается, но это не учебник. ActionScript подробно рассмотрен в массе хороших книг, справочников и документов. «ActionScript 3.0. Сборник рецептов» преследует иные цели. Особый формат этой книги призван помочь читателям в решении конкретных задач.

Совместимость

Книга называется «ActionScript 3.0. Сборник рецептов». То есть все примеры кода в ней базируются на ActionScript 3.0 и совместимы только с продуктами, поддерживающими ActionScript 3.0. Flex 2.0 и Flash 9 позволяют писать на ActionScript 3.0. Flash Player 9 поддерживает ActionScript 3.0. Если продукт, с которым вы работаете, не поддерживает ActionScript 3.0, то код, приведенный в данной книге, скорее всего функционировать не будет.

Принятые наименования платформы Flash

ActionScript 3.0 – важная часть Flash Platform, включающей множество взаимосвязанных технологий, ориентированных на Flash Player. Сегодня технологий Flash Platform так много, что отследить их все очень сложно. Еще более усугубляет ситуацию неформальное, неаккуратное и даже неправильное употребление терминов многими разработчиками. Мы не претендуем на роль блюстителей чистоты терминологии, но стремимся к максимальной точности и ясности при рассмотрении этих технологий в данной книге и поэтому будем оперировать терминами весьма конкретно. В табл. 1 собраны употребляемые нами названия и их значения.

Таблица 1. Принятые в данной книге наименования Flash Platform

Имя	Значение
Flex framework	Библиотека классов ActionScript, поставляемая как часть Flex SDK и Flex Builder.
Flex Builder	Интегрированная среда разработки Adobe для создания Flex-приложений.
Flex SDK	Компилятор и инфраструктура Flex, используемая для создания Flex-приложений без Flex Builder.
Flex	Технология, применяемая для создания SWF-файлов из документов MXML и файлов ActionScript. Если не указан номер версии, речь идет о Flex 2.0.
Flash Player	Платформа для работы SWF-файлов, созданных во Flash или Flex. Если не указан номер версии, речь идет о Flash Player 9.
Flash	Среда разработки, используемая для создания SWF-файлов. Если не указан номер версии, речь идет о Flash 9.*

* Официально продукт называется Flash CS3. – *Примеч. науч. ред.*

Код

В данной книге много ActionScript 3.0 (попросту говоря, кода). Во многих рецептах предлагаются специальные классы, которые, на наш взгляд, бесценны. Уверены, что и вы найдете их полезными.

Скачать библиотеки ActionScript 3.0 полностью можно по адресу *http://www.rightactionscript.com/ascb*. После загрузки файлы библиотеки надо добавить в путь к классам ваших проектов. Инструкции по настройке пути к классам ActionScript 3.0 также можно найти на этом сайте.

Аудитория книги

Подходит ли вам эта книга? Конечно, мы надеемся, что да. Но чтобы исключить сомнения, сделаем краткий обзор целей, которые она преследует, и знаний, необходимых для работы с ней.

Что надо знать

Предполагается, что читатель уже хорошо знаком с продуктом или продуктами, которые применяет для создания информационного наполнения Flash Player. Здесь не рассматриваются основы Flex SDK, Flex Builder или Flash. Данная книга будет вам полезнее, если вы сначала освоите азы работы с этими продуктами, в частности научитесь компилировать и развертывать проект.

Кроме того, прежде чем читать эту книгу, не помешает узнать базовые принципы программирования. Мы много говорим об основных приемах программирования в контексте решения конкретных задач (например, как перебрать все элементы массива), но все-таки это не подробное руководство по базовым навыкам программирования.

Для кого написана эта книга

Для всех разработчиков на ActionScript 3.0. Надеемся, что все – от новичка до эксперта – откроют в ней что-то полезное для себя. Она идеально подходит для тех, кому надо быстро найти решение.

Кому эта книга ни к чему

Данная книга посвящена ActionScript 3.0 – языку программирования, который выполняется во Flash Player. По большей части в ней не обсуждаются подробно решения, работающие на серверной стороне, или другие языки программирования, которые могут применяться на стороне клиента. Так, здесь рассматривается код на ActionScript для работы с Flash Remoting (технологией для осуществления удаленных вызовов процедур) и приводятся примеры, но не объясняется, как писать соответствующий код, работающий на стороне сервера (например, компонент ColdFusion). Аналогично обсуждается применение Action-

Script для вызова функций JavaScript, но не рассказывается, как писать JavaScript. Если вы ищете книгу по ActionScript 3.0, то она перед вами. Но подробное обсуждение тем, выходящих за рамки ActionScript, надо искать в других источниках.

Структура книги

Книга состоит из двадцати четырех глав и одного приложения.

Глава 1 «Основы ActionScript»

Базовые понятия программирования, такие как операторы цикла, таймеры и т. д.

Глава 2 «Специальные классы»

Написание специальных классов для ActionScript 3.0.

Глава 3 «Среда времени выполнения»

Получение информации об операционной системе, устройстве и версии проигрывателя, а также о системе безопасности.

Глава 4 «Числа и математические операции»

Работа с числами в ActionScript, включая синтаксический разбор из строк, преобразование в форматированные строки и применение различных систем счисления.

Глава 5 «Массивы»

Работа с индексированными наборами данных, которые называют массивами, – от добавления и удаления элементов до сортировки.

Глава 6 «Список отображения»

Применение отображаемых объектов для вывода визуальных данных на экран.

Глава 7 «Программное создание изображений и масок»

Программное создание изображений и масок с помощью ActionScript.

Глава 8 «Растровые изображения»

Низкоуровневая работа с растровыми изображениями.

Глава 9 «Текст»

Все о тексте – от отображения до загрузки и форматирования.

Глава 10 «Фильтры и трансформации»

Применение различных эффектов к отображаемым объектам с помощью трансформаций (цветовых и геометрических) и фильтров, таких как тени, фаски и даже эффекты тиснения и выделения краев.

Глава 11 «Программная анимация»

Анимирование отображаемых объектов с помощью ActionScript.

Глава 12 «Строки»

Работа со строковыми данными – от поиска подстрок до работы с Unicode.

Глава 13 «Регулярные выражения»

Создание собственных регулярных выражений для сопоставления шаблонов и строк.

Глава 14 «Даты и время»

Работа с датами и временем, включая преобразования между часовыми поясами с помощью таймеров и форматирование дат.

Глава 15 «Программирование звука»

Работа с аудиоданными, включая загрузку MP3, чтение тегов ID3 и отображение звуковых волн.

Глава 16 «Работа с видеоданными»

Программирование для Flash-видео.

Глава 17 «Хранение данных»

Применение совместно используемых объектов для хранения данных на клиентском компьютере.

Глава 18 «Обмен информацией между роликами»

Использование локальных соединений для обмена данными между содержимым, выполняющимся в экземпляре Flash Player на одном компьютере.

Глава 19 «Отправка и загрузка данных»

Организация двунаправленного обмена данными между веб-сервером и Flash Player.

Глава 20 «XML»

Работа с XML с использованием поддержки E4X во Flash Player.

Глава 21 «Веб-сервисы и удаленное взаимодействие во Flash»

Работа с вызовами удаленных процедур, основанная на технологиях веб-сервисов и Flash Remoting.

Глава 22 «Создание интегрированных приложений»

Использование программного интерфейса Flash Player для интегрирования его содержимого с хост-приложением, например для вызова функций JavaScript из ActionScript или для вызова функций ActionScript из JavaScript.

Глава 23 «Работа с файлами»

Передача и загрузка файлов.

Глава 24 «Работа с сокетами»

Применение сетевых соединений для передачи XML и двоичных данных в приложениях с малым периодом ожидания.

Приложение «Таблица кодов Unicode для символов Latin 1»

В приложении приводится список кодов для кодировки Latin 1 с эквивалентными им Unicode-кодами в диапазоне от U+0000 до U+00FF (т. е. управляющих символов C0, базовых символов Latin, управляющих символов C1 и дополнительных символов Latin 1).

Как работать с этой книгой

Пусть она будет вашим другом и советчиком. Не ставьте ее на полку. Положите на свой рабочий стол, чтобы как можно чаще обращаться к ней за помощью. Засомневавшись, что понимаете, как работает тот или иной элемент кода или как решить задачу, возьмите эту книгу и найдите нужный(ые) рецепт(ы). Ее формат призван помочь читателю быстро получить ответы на свои вопросы. А поскольку это книга, то она никогда не посмеется над вами и вашими вопросами. А важных или неважных вопросов не бывает.

Ее можно прочитать от корки до корки, но мы советуем обращаться к ней, когда понадобится ответ на конкретный вопрос. Она не будет мучить вас лекциями по теории, а постарается помочь решить поставленные задачи. Эта книга для полевых работ, а не для исследовательской лаборатории.

Типографские обозначения

В данной книге приняты следующие типографские обозначения:

Обычный текст

Показывает заголовки меню, опции меню, кнопки меню и «быстрые» клавиши (такие как Alt и Ctrl)

Курсив

Показывает новые термины, URL, адреса электронной почты, имена файлов, расширения файлов, имена путей, каталоги и утилиты UNIX.

`Моноширинный`

Показывает команды, параметры, переключатели, переменные, атрибуты, ключи, функции, типы, классы, пространства имен, методы, модули, свойства, параметры, значения, объекты, события, обработчики событий, теги XML и HTML, макросы, содержимое файлов или результат выполнения команд.

`Моноширинный полужирный`

Показывает команды или другой текст, который должен быть введен пользователем как есть.

Моноширинный курсив

Показывает текст, который должен быть заменен пользовательскими значениями.

При описании свойств или методов объектов и классов действуют следующие обозначения:

- При записи констант уровня класса как имя класса, так и свойство записываются моноширинным шрифтом, потому что они должны вводиться «как есть», например `Event.COMPLETE`.
- Для свойств уровня экземпляра экземпляр класса или объекта записывается *моноширинным курсивом*, потому что они должны быть заменены конкретным экземпляром. Само свойство записывается моноширинным шрифтом и должно вводиться без изменений, например *`Button.enabled`*.
- Имена методов и функций, а также класс или объект, к которым они относятся, всегда выделяются курсивом и оканчиваются круглыми скобками: *BitmapData.clone( )*. В оперативной справке можно узнать, включать ли имя класса (т. е. относится ли метод к так называемым статическим методам), как в *String.fromCharCode( )*, или заменять его именем экземпляра, как в *exampleBitmap.clone( )*.
- Для краткости при обсуждении свойства или метода мы всегда опускаем имя его класса. Например, когда речь идет о свойстве `htmlText` класса `TextField`, если говорится «зададим свойство `htmlText`», вы должны по контексту понимать, что имеется в виду «зададим свойство *`exampleTextField`*`.htmlText`, где *`exampleTextField`* — это идентификатор конкретного текстового поля».

Кроме того, чтобы привлечь внимание читателя и оживить изложение, в текст встроены следующие примечания:

Это подсказка, предложение или совет общего характера.

Это предупреждение или предостережение.

Примеры кода

«ActionScript 3.0. Сборник рецептов» призван помочь разработчику в его труде. Вы можете заимствовать код из этой книги и поместить его в свои программы и документацию. Если в программе используется несколько блоков кода из этой книги, то обращаться к нам за разрешением не надо. А вот продажа или распространение CD-ROM с примерами из книг O'Reilly *требует* специального разрешения. Вы можете

свободно ссылаться на книгу и цитировать примеры кода, но для включения больших фрагментов кода из этой книги в документацию вашего продукта *требуется* наше согласие.

Будем благодарны, но не настаиваем на указании авторства. Обычно ссылка на источник включает название, автора, издателя и ISBN. Например: «ActionScript 3.0 Cookbook, by Joey Lott, Darron Schall, and Keith Piters. Copyright 2007 O'Reilly Media, Inc., 978-0-596-52695-5».

Если вам кажется, что использование вами примеров кода выходит за рамки, оговоренные выше, не стесняйтесь обратиться к нам по адресу *permissions@oreilly.com*.

Сборники рецептов O'Reilly

Где взять рецепт решения программистской задачи? Возьмите сборник от O'Reilly, и больше вам ничего не понадобится. В каждом из них вы найдете сотни сценариев, программ и последовательностей команд, которые вы можете использовать для решения конкретных задач.

Общая формула рецептов из этого сборника такова:

Задача

Все задачи, рассматриваемые здесь, четко сформулированы, конкретны и реальны.

Решение

Решение доступно для понимания и реализации.

Обсуждение

Обсуждение разъясняет суть задачи и решения. В нем также приводится пример кода, показывающий, как сделать все наилучшим образом. А самое замечательное, что все примеры, приведенные в сборнике, можно скачать с веб-сайта книги по адресу *http://www.oreilly.com/catalog/actscpt3ckbk*.

См. также

Раздел «См. также» отсылает читателя к дополнительной информации по рассматриваемой в рецепте теме – другим рецептам, книгам (включая изданные не O'Reilly), сайтам и т. д.

Дополнительную информацию о серии «Сборники рецептов O'Reilly» можно найти на сайте *http://cookbooks.oreilly.com*.

Safari® Enabled

Если на обложке книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи

лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Как с нами связаться

Комментарии и вопросы издателю по этой книге направляйте по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в Соединенных Штатах или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

У этой книги есть веб-страница, на которой перечислены опечатки, примеры и другая дополнительная информация. Ее можно найти по адресу:

http://www.oreilly.com/catalog/actscpt3ckbk

Авторы создали сайт этой книги по адресу:

http://www.rightactionscript.com/ascb

Чтобы прокомментировать или задать технические вопросы по книге, присылайте электронные письма по адресу:

bookquestions@oreilly.com

Для получения дополнительной информации о наших книгах, конференциях, информационных центрах и O'Reilly Network посетите наш веб-сайт по адресу:

http://www.oreilly.com

О научных редакторах

Стейси Малкахи (Stacey Mulcahy) – разработчик во Flex и Flash, у которой однажды возник роман с Macromedia Director. Когда она не создает многофункциональные интернет-приложения, которые нравятся даже дизайнерам, ее можно найти в учебных классах, где в качестве инструктора по Flash она учит других делать то же самое. Она властвует на своем блоге *http://www.bitchwhocodes.com*, посвященном Flash.

Сэм Роббинс (Sam Robbins) работает с Flash и ActionScript более шести лет и почти два года экспериментирует с Flex. Все это время основные свои силы он направляет на разработку многофункциональных интернет-приложений. Кроме того, он организатор подгруппы в Бостонской группе пользователей Flash Platform (Boston Flash Platform User Group, BFPUG), которая ежемесячно тестирует шаблоны проектирования.

В свободное время Сэм пытается вести свой блог (*http:// pixelconsumption.com*), играет в Xbox и стрижет газон. Он живет в Массачусетсе.

Стивен Шелтер (Steven Schelter) – программист, в настоящее время работающий в Schematic. Прекрасно владеет различными языками сценариев и программирования серверной и клиентской частей, но основное его занятие – разработка на ActionScript. Стивен перешел в индустрию интерактивных систем, поскольку Flash позволил сочетать его дизайнерские способности со знаниями математики и логистики. Он также интересуется 3D-моделированием, искусственным интеллектом и интерактивными пользовательскими приложениями.

Роджер Браунштейн (Roger Braunstein) – разработчик и дизайнер из Бруклина. Одержим красивым кодом, разработкой анимации, программированием графических объектов и видеоиграми. Также интересуется кулинарией, фотографией, велосипедами и 8-разрядной музыкой. Он надеется, что когда-нибудь у него будет настоящий сайт по адресу *http://www.partlyhuman.com*.

Мюон Ти Ван (Muon Thi Van) – разработчик программного обеспечения из компании Schematic, интерактивного агентства полного цикла, имеющего офисы в Лос-Анджелесе и Нью-Йорке. Она является архитектором, разработчиком и дизайнером множества Flash-приложений и игр для Всемирной паутины и мобильных устройств, выступает на профессиональных конференциях, таких как Flashbelt и Flashforward. Мюон получила степень бакалавра по вычислительной технике в Северо-Западном университете.

Дэниел Вильямс (Daniel Williams) – разработчик программного обеспечения компании Schematic, Нью-Йорк, где живет и дышит ActionScript. Он даже видит его во сне. Кроме этого он активно интересуется физикой, быстрыми двухколесными транспортными средствами и человеческим мозгом. Время от времени делится своими мыслями и опытом на своем сайте с неподходящим названием *http://www.danieldoesdallas.com*.

Благодарности

Эта книга представляет собой результат работы трех авторов, Джои Лотта, Деррона Шалла и Кейта Питерса, чьи имена вы видите на обложке. Однако, безусловно, книга бы не состоялась без помощи многих людей, которые не значатся среди авторов. Мы хотели бы выразить им нашу коллективную признательность.

Спасибо Чаку Топореку (Chuck Toporek) за его преданность, терпение и фантастическое редактирование.

Также спасибо Стиву Вейсу (Steve Weiss) за веру в книгу и ее авторов. Мы благодарны за неизменно полезные и позитивные советы.

Спасибо Тиму О'Рейли (Tim O'Reilly) за высокую планку в технической литературе и за то, что сделал возможным выход этой книги в таком формате. Для нас было честью работать с такой прогрессивной и передовой компанией.

Без работы всего редакторского и технического состава O'Reilly эта книга не была бы там, где она находится сейчас. Спасибо всем, кто вложил в нее свое время и энергию.

Мы хотели бы поблагодарить нашего агента Марго Хатчинсон (Margot Hutchinson) из Waterside Productions за помощь в согласовании всех необходимых деталей.

Помощь команд Flex, Flash и Flash Player из Adobe всегда бесценна и чрезвычайно полезна. Они отвечают на наши вопросы, а мы можем ответить на ваши. Спасибо всем в Adobe.

Также хотим выразить благодарность всем научным редакторам, при содействии которых эта книга обрела лучшую форму, какая только возможна.

Джои Лотт

Спасибо вам, Кейт и Деррон, за помощь в создании книги. Для меня было честью работать с двумя самыми выдающимися экспертами в этой области.

Я также хотел бы выразить признательность моим друзьям и семье за их поддержку и энтузиазм. Я благодарен судьбе за все.

Деррон Шалл

Спасибо тебе, Джои, за то, что предоставил мне возможность заполнить эти страницы. Кейт, и тебе спасибо за совместную работу. Писать с вами было просто замечательно, и я горд быть вашим соавтором. Чак и Стив, ваша поддержка неоценима. Я, несомненно, не справился бы со всем этим без вас.

Спасибо моей красавице жене Джен за понимание и бесконечное терпение. Я люблю тебя.

Всей моей семье, особенно моему деду Эдвину, спасибо за вашу поддержку и веру в меня.

Кейт Питерс

Спасибо тебе, Джои, за возможность стать частью этого проекта. Также благодарю Стива, Чака и Деррона за помощь в ходе работы и, как обычно, Казуми и Кристин за то, что слишком часто им приходится терпеть меня, уткнувшегося в монитор.

20

XML

20.0. Введение

XML – это средство структурированного форматирования и описания данных в текстовом представлении. Изначально этот язык разметки разрабатывался для обеспечения простоты и гибкости, и благодаря своей универсальности, особенно в применении к обмену данными и обеспечению возможности взаимодействия приложений, быстро превратился в промышленный стандарт.

С XML можно столкнуться и при работе с ActionScript. В главе 19 рассказывается, как посылать и загружать данные в URL-кодированном формате. URL-кодирование годится для обмена простыми данными между Flash Player и сценариями, выполняющимися на стороне сервера, но для сложных данных или символов Unicode лучше использовать XML, поскольку он структурирован. Например, из текстового файла требуется загрузить простой тип данных, такой как строка. С помощью экземпляра *URLLoader* могут быть загружены такие URL-кодированные данные:

```
myString=a+string+value
```

Однако если требуется загрузить данные из внешнего источника и использовать их для создания объекта ActionScript, возникает проблема, как представить эти данные в виде URL-кодированной строки. Можно придумывать различные приемы, как например здесь, где каждая пара свойство-значение выделена звездочками (`*`) и каждое свойство отделено от его значения вертикальной чертой (`|`):

```
myObject=prop0|val0*prop1|val1*prop2|val2
```

Получив строковое значение `myObject` (мой объект), с помощью свойства *String.split( )* можно воссоздать элементы, образующие объект. Та-

кой подход имеет право на жизнь, однако, как правило, сложные значения намного проще представлять в формате XML. Например, тот же объект можно представить следующим фрагментом XML:

```
<myObject>
  <prop0>val0</prop0>
  <prop1>val1</prop1>
  <prop2>val2</prop2>
</myObject>
```

XML-данные предлагают ряд преимуществ по сравнению с URL-кодированными данными:

- С помощью XML намного проще представлять сложные данные. XML можно создавать вручную (для статического XML-документа) или программно (из сценария ColdFusion, PHP и т. д.).
- Большинство языков серверных сценариев предлагают встроенную функциональность для чтения и генерирования XML-данных.
- XML – это стандарт, используемый для передачи и хранения данных во всех типах приложений и платформ.

Конечно, XML не единственный способ передачи данных для Flash Player. В главах 19, 21 и 24 обсуждаются средства передачи данных и вне Flash Player. Однако данная глава посвящена исключительно XML, стандартизованной технике обмена данными, которая не требует применения дополнительного серверного ПО (в отличие от Flash Remoting и сетевых соединений). XML стал важной частью ActionScript 3.0, в котором появляется специальное средство работы с XML.

ActionScript 3.0 может похвастать революционно новым синтаксисом для работы с XML. Спецификация *ECMAScript for XML, которой соответствует* ActionScript 3.0, известная также как *E4X*, – это расширение языка *ECMAScript*, обеспечивающее более простой и удобный для чтения подход для работы с объектами XML, чем традиционная, безнадежно устаревшая *объектная модель документов* (Document Object Model, DOM). С появлением E4X работать с XML стало намного проще, чем раньше. Кроме того, если вы впервые сталкиваетесь с XML, E4X сильно ускоряет процесс обучения работе с ним.

В данной главе мы придерживаемся следующей терминологии:

документ XML

Файл, содержащий XML. Данным термином также могут обозначаться загружаемые или отправляемые XML-данные. Не надо путать документ XML с классом *XMLDocument*.

пакет XML

Пакетом XML может быть любой фрагмент XML – от целого документа XML до отдельного узла – если он представляет завершенный, корректный (well-formed) элемент информации в формате XML.

узел XML

Основной строительный блок XML. Узлами могут быть элементы, текстовые узлы, атрибуты и т. д. Обычно элементы и текстовые узлы называют просто «узлами».

элемент XML

Термин «элемент» обычно употребляют как синоним термина «тег». Однако если быть более точным, элемент содержит теги. Элементы должны иметь открывающий и закрывающий теги (`<элемент></элемент>`), или открывающий и закрывающий теги могут быть объединены в один, если элемент не содержит вложенных элементов (`<элемент />`).

Корневой узел

Элемент, находящийся в вершине XML-иерархии элементов.

Текстовый узел

Узел, содержащий текст. Текстовые узлы обычно вложены в элементы.

Атрибут

Это часть элемента. Атрибуты размещаются в тегах элементов в формате имя/значение, например `<элемент имя="значение">`.

объявление XML

Объявление обычно выглядит так: `<?xml version="1.0" ?>`. Это специальный тег, который распознается анализатором XML как тег, содержащий информацию об XML-документе, и не разбирается как элемент.

дерево XML

Иногда его также называют «дерево данных». Дерево XML – это иерархия узлов в XML-документе.

20.1. Изучение структуры XML (чтение и запись XML)

Задача

Научиться понимать, как записывать и читать XML.

Решение

XML основывается на тегах и имеет иерархическую структуру. Если вы знаете HTML, изучение основ XML не должно вызвать трудностей.

Обсуждение

Хотя при работе с ActionScript чтение и написание хорошего XML не главное, но от этого ваш ActionScript только выиграет. Если вы еще не знакомы с XML, не волнуйтесь, в нем нет ничего страшного.

XML – это способ представления структурированных данных. То есть содержимое данных описывается явно. Например, есть строка данных:

```
Джерри, Кэролин, Лаура
```

С помощью XML можно сообщить, *кто* эти люди:

```
<family>
  <father>Джерри</father>
  <mother>Кэролин</mother>
  <sister>Лаура</sister>
</family>
```

Теперь, как видите, XML предоставляет намного больше информации о данных. Вот еще несколько моментов, которые необходимо отметить об XML:

- Структура документа XML образована главным образом узлами. Узел – это общий термин, который применим ко многим частям XML. Например, `<family>` (семья) – это узел предыдущего фрагмента XML; такие узлы называют *элементами*. Значения `Джерри`, `Кэролин` и `Лаура` тоже представляют собой узлы; такие узлы называют *текстовыми*.
- Каждый элемент XML должен иметь соответствующие открывающий и закрывающий теги. Пример открывающего тега: `<family>`. Закрывающий тег идентичен открывающему, за исключением наличия в нем прямого слэша, обозначающего, что это закрывающий элемент: `</family>`. Если элемент не содержит вложенные узлы, открывающий и закрывающий теги могут быть объединены. Например, `<emptyElement />` (пустой элемент) – элемент, сочетающий в себе открывающий и закрывающий теги. Обратите внимание на пробел между именем элемента, `emptyElement`, и прямым слэшем. Пробел может и отсутствовать – это дело вкуса.
- Элементы могут содержать вложенные узлы (будь то другие элементы или текстовые узлы). Несколько примеров этому можно найти в приведенном выше фрагменте XML-документа `<family>`. Элемент `<family>`, который является *корневым узлом* в данном примере, содержит три вложенных элемента: `<father>` (отец), `<mother>` (мать) и `<sister>` (сестра). Такие вложенные узлы еще называют *дочерними узлами*. Каждый из этих дочерних узлов также содержит вложенный узел. Однако эти вложенные узлы являются текстовыми узлами, а не элементами. Несмотря на это они по-прежнему рассматриваются как дочерние узлы.

Нам хотелось бы рассмотреть здесь еще один тип узлов. *Атрибут* – это специальный тип узла, который задается элементу и во многих случаях может даже использоваться как альтернатива вложенному узлу. Если вы когда-нибудь работали с HTML, то уже знакомы с атрибутами. Среди самых распространенных атрибутов HTML можно назвать атрибут `href` элемента `<a>` и атрибут `colspan` элемента `<td>`. Вот как из-

менится рассматриваемый ранее XML-документ, если заменить в нем вложенные узлы атрибутами:

```
<family father="Джерри" mother="Кэролин" sister="Лаура" />
```

Обратите внимание, что мы смогли убрать вложенные элементы и записать те же данные в одном элементе. Также заметьте, что поскольку в <family> больше нет вложенных узлов, открывающий и закрывающий теги можно объединить.

Возникает вопрос, когда и почему использовать атрибуты, а когда – вложенные узлы. Часто это дело вкуса. Иногда проще или понятнее выглядит запись данных XML с помощью атрибутов. Как правило, если необходимо представить сравнительно небольшое количество относительно коротких значений, лучше выбрать атрибуты. Имена атрибутов должны быть уникальными в рамках элемента. Если требуется представить много данных, если данные довольно длинные (более нескольких слов) или если невозможно обеспечить уникальность имен атрибутов в элементе, используются вложенные элементы.

Атрибуты и вложенные узлы можно сочетать. Вот пример элемента <article> (статья), который включает атрибуты title и author, но использует вложенный текстовый узел для представления текста статьи. Это хороший пример того, когда одно из значений (текст статьи) просто слишком длинное, чтобы быть атрибутом.

```
<article title="XML: не только для чокнутых" author="Сэмюел Р.
Шимовитц">
Мои друзья не могли поверить в то, что я занялся XML.
Я стал изгоем, заключенным в темноте офиса, освещаемого только моим верным
монитором.
</article>
```

При составлении XML-данных для отправки во внешний сценарий XML-строки можно создавать прямо во Flash. Например:

```
var dataToSend:String = "<feedback name='Анон' comments='хорошо'>";
```

Статический документ XML вне Flash создается в любом текстовом редакторе и сохраняется как простой текст. Динамическое создание XML-документа с использованием серверного сценария можно найти в соответствующей документации языка программирования.

См. также

В данном рецепте приведены очень простые примеры. XML может быть намного более сложным и включать пространства имен, объявления типов документов и т. д. Большинство из этих элементов выходят за рамки данной книги и за рамки того, что требуется знать для работы с XML в ActionScript. Более подробную информацию об XML можно найти на сайте *http://www.xml.com*. Чтобы получить общее представление об XML, рекомендуем обратиться к книге Эллиотта Расти

Гарольда (Elliotte Rusty Harold) и В. Скотта Минса «XML in a Nutshell» (O'Reilly)[1] и/или книге Саймона Ст. Лорента (Simon St.Laurent) и Майкла Фицджеральда (Michael Fitzgerald) «XML Pocket Reference» (O'Reilly).

20.2. Создание объекта XML

Задача

Требуется создать объект XML с древовидной структурой и данными.

Решение

Заполненный XML-объект содержит данные, как город, заселенный людьми.

Заполненный XML-объект можно создать одним из следующих способов:

- Создайте объект XML и заполните его, присвоив ему XML-литерал.
- Заполните дерево объекта XML, передавая в конструктор *XML* строку XML.
- Создайте пустой объект XML и с помощью E4X заполняйте узлы дерева XML под одному.
- Создайте пустой объект и загрузите в него XML-данные из внешнего источника.

Обсуждение

Заполненный объект XML в ActionScript можно создавать по-разному. Каждый метод предлагает свои преимущества, поэтому при выборе необходимо исходить из нужд конкретного проекта.

Самый простой способ создать заполненный XML-объект – создать новый XML-объект и присвоить ему XML-литерал:

```
var example:XML = <abc><a>eh</a><b>bee</b><c>see</c></abc>;
```

Это пример E4X в действии. Обратите внимание, что пакет XML используется прямо в коде в правой части равенства. Если бы XML был заключен в кавычки, он интерпретировался бы как строка, но поддержку типов данных XML встроена в ActionScript 3.0. Компилятор ActionScript понимает, что выражение в правой части – это XML, и заполняет XML-экземпляр `example` предоставленным пакетом XML.

Такая техника подходит, если структура XML известна еще до создания объекта. Она даже допускает наличие динамических данных в ли-

[1] Гарольд Э., Минс С. «XML. Справочник». – Пер. с англ. – СПб: Символ-Плюс, 2002.

терале XML. Имя переменной в литерале XML заключается в фигурные скобки ({and }). Например, для простого XML-объекта, предназначенного для передачи на сервер имени пользователя и счета, был бы создан такой XML-литерал:

```
// Предположим, существует две переменные,
// username и score
var username:String = "Деррон";
var score:int = 1000;

// Заключаем имя переменной в фигурные скобки,
// чтобы использовать ее значение при создании
// объекта XML с помощью XML-литерала
var example:XML = <gamescore>
                    <username>{username}</username>
                    <score>{score}</score>
                  </gamescore>;
```

Также можно создать строку и передать ее как параметр в конструктор XML. Предыдущий пример тогда записывается следующим образом:

```
// Создаем структуру XML с помощью строки с использованием
// значений username и score в пакете XML.
var str:String = "<gamescore><username>" + username + "</username>"
               + "<score>" + score + "</score></gamescore>";

// Передаем строку в конструктор для создания объекта XML
var example:XML = new XML( str );
```

Структура XML-объектов `example`, созданных в двух предыдущих примерах кода, идентична. XML-объекты можно создавать с использованием фигурных скобок, если на момент компиляции известна их структура, но данные поставляются во время выполнения (как счет, который формируется после окончания игры).

Иногда не все данные известны сразу. В этом случае и дерево объекта XML создается лишь через некоторое время. Так, если XML-объект предназначен для хранения информации о корзине покупок пользователя, данные объекта должны меняться при каждом изменении набора товаров в корзине. В таких случаях создается объект XML, в который затем с помощью E4X добавляются или удаляются узлы, как описывается в рецептах 20.3 и 20.10.

И не будем забывать о заполнении объектов XML данными, извлеченными из внешнего источника, такого как статический XML-документ или сценарий, который генерирует динамический XML. До сих пор мы рассматривали создание XML-объектов только для отправки данных на сервер или для использования в собственных программах, но можно и загружать данные с сервера, например при извлечении пользовательских данных, информации каталога или настроечной информации ролика. Для таких сценариев в рецепте 20.11 рассказывается, как загружать документ XML с сервера.

См. также

Рецепты 20.3, 20.4, 20.5 и 20.10, в которых более подробно рассказывается о создании дерева XML с использованием E4X. О загрузке информации из внешних источников рассказывает рецепт 20.11.

20.3. Добавление элементов в объект XML

Задача

Требуется создать объект *XML* и добавить в него элементы.

Решение

Используйте синтаксис E4X для создания дочерних элементов и добавления их в дерево XML. Кроме того, больший контроль за добавлением новых элементов обеспечат методы *insertChildBefore( )* и *insertChildAfter( )*.

Обсуждение

В объект XML можно добавлять элементы для создания структуры данных XML и ее передачи в другое приложение. Это можно сделать несколькими способами, как обсуждается в рецепте 20.2.

Синтаксис E4X позволяет без труда добавлять элементы в объект XML. Элементы добавляются в экземпляр XML с помощью оператора «точка» (.), практически аналогично добавлению типичного свойства в обычный *Object*:

```
// Создаем экземпляр XML, в который будем добавлять элементы
var example:XML = <example />;

// Создаем новый узел XML newElement
// и добавляем его в экземпляр example
example.newElement = <newElement />;

/* На экран выводится:
  <example>
    <newElement/>
  </example>
*/
trace( example );
```

В предыдущем примере создается литерал XML для пустого узла `newElement` (новый элемент), который затем добавляется в экземпляр `example` посредством оператора «точка» и присвоения свойства `newElement` узлу `newElement`. В данной ситуации не обязательно, чтобы имя свойства и имя элемента совпадали. Однако было бы странно, если бы имена были разными. При добавлении узла XML в существующий объект XML имя элемента замещает имя свойства.

Также новый элемент можно ввести, создавая свойство экземпляра XML и присваивая ему значение:

```
// Создаем экземпляр XML
var example:XML = <example />;

// Создаем новое свойство emptyElement
// и присваиваем ему пустую строку
example.emptyElement = "";

/* На экран выводится:
  <example>
    <emptyElement/>
  </example>
*/
trace( example );
```

Предыдущий пример демонстрирует, что дочерние элементы XML-объекта можно создавать, не добавляя XML-узлы фактически, как в первом примере. Задавая пустую строку как значение свойства `emptyElement` (пустой элемент), мы создаем пустой узел элементов `emptyElement`.

Если имя добавляемого элемента неизвестно или если оно должно основываться на значении переменной, можно использовать альтернативную запись с квадратными скобками. Вместо применения оператора «точка» заключаем строку в квадратные скобки (`[` и `]`). Строка вычисляется и полученное значение выступает в качестве имени свойства. Это позволяет динамически создавать имя свойства и, следовательно, имя элемента:

```
// Создаем экземпляр XML
var example:XML = <example />;

var id:int = 10;

// Создаем строку, чтобы включить значение id в имя узла
example[ "user" + id ] = "";

/* На экран выводится:
  <example>
    <user10/>
  </example>
*/
trace( example );
```

Иногда без синтаксиса квадратных скобок не обойтись. Например, в XML в имени узла элементов допускается дефис. Попытка создать имя элемента, включающее дефис, с помощью оператора «точка» приводит к ошибке компиляции:

```
example.some-element = ""; // Формируется ошибка компиляции
```

Дефис имеет специальное значение для компилятора: он является знаком вычитания. Предыдущий фрагмент кода интерпретируется как

выражение вычитания переменной *element* из *example.some* и последующего присваивания пустой строки этому значению. Такое выражение вычитания не является синтаксически правильным, что объясняет появление ошибки компилятора. Более того, даже если бы это выражение было скомпилировано, оно бы не обеспечило необходимого поведения создания узла элементов. Применение квадратных скобок устраняет сложности:

```
example[ "some-element" ] = "";
```

Простота E4X, однако, имеет и неприятную оборотную сторону: дело в том, что новый элемент всегда добавляется только в конец дерева XML. Методы *insertChildBefore( )* (вставить дочерний элемент перед) и *insertChildAfter( )* (вставить дочерний элемент после) обеспечивают больший контроль над добавлением новых узлов. Первый из них вставляет новый элемент перед элементом дерева XML, а второй – после элемента. Оба метода принимают два одинаковых параметра: узел, отмечающий точку вставки, и вставляемые данные. Следующий фрагмент кода иллюстрирует изменение дерева XML с помощью обоих методов:

```
// Создаем экземпляр XML
var example:XML = <example/>;

// Создаем пустой узел элемента two
example.two = "";

// Вставляем узел one перед узлом two
example = example.insertChildBefore( example.two, <one /> );

// После узла two вставляем узел three
example = example.insertChildAfter( example.two, <three /> );

/* На экран выводится:
<example>
  <one/>
  <two/>
  <three/>
</example>
*/
trace( example );
```

Обратите внимание, что методы *insertChildBefore( )* и *insertChildAfter( )* не меняют экземпляр XML, для которого они вызваны, а возвращают новый экземпляр, содержащий изменения. Чтобы зафиксировать изменения, результат вызова метода необходимо опять присвоить экземпляру XML, как показано выше.

См. также

Рецепты 20.2 и 20.4.

20.4. Добавление текстовых узлов в объект XML

Задача

Требуется добавить текстовые узлы в объект *XML*.

Решение

Используя синтаксис E4X, создайте текстовые узлы и добавьте их в дерево XML. Для получения большего контроля над добавлением текстовых узлов обратитесь к методам *appendChild( )*, *prependChild( )*, *insertChildAfter( )* и *insertChildBefore( )*.

Обсуждение

Создавать текстовые узлы с помощью E4X, как вы уже догадались, просто. Процесс аналогичен тому, который был описан в рецепте 20.3. Как и в случае узлов элементов, для создания текстовых узлов используется оператор «точка» (.), с помощью которого создается свойство экземпляра XML и ему присваивается любое значение, которое может быть преобразовано в строку:

```
// Создаем экземпляр XML
var example:XML = <example/>;

// Создаем текстовый узел из строки
example.firstname = "Деррон";

// Создаем текстовый узел из числа
example.number = 24.9;

// Создаем текстовый узел из Булева значения
example.boolean = true;

// Создаем текстовый узел из массива
example.abc = ["a", undefined, "b", "c", null, 7, false];

/* На экран выводится:
<example>
  <firstname>Деррон</firstname>
  <number>24.9</number>
  <boolean>true</boolean>
  <abc>a, , b, c, , 7, false</abc>
</example>
*/
trace( example );
```

В этом примере в XML-экземпляр добавляются и текстовые узлы, и узлы элементов. Значение в правой части выражения присваивания преобразуется в строку и становится текстовым узлом. Затем имя свойства преобразуется в узел элемента с текстовым узлом в качестве дочернего элемента, и весь узел элемента добавляется в экземпляр XML.

Для управления размещением текстовых узлов в узлах элементов используются методы *appendChild( )* (добавить дочерний элемент в конец), *prependChild( )* (добавить дочерний элемент в начало), *insertChildBefore( )* или *insertChildAfter( )*. Эти методы позволяют точно определить, как текстовый узел вставляется в дерево XML:

```
// Создаем экземпляр XML
var example:XML = <example/>;

// Добавляем в конец дерева узел элемента two, содержащий
// дочерний текстовый узел со значением 2
example.appendChild( <two>2</two> );

// Добавляем в начало дерева узел элемента one, содержащий
// дочерний текстовый узел со значением "номер 1"
example.prependChild( <one>"номер 1"</one> );

// После узла элемента one вставляем текстовый узел
// со значением 1.5
example.insertChildAfter( example.one[0], 1.5 );

// Перед узлом элемента two вставляем узел элемента part,
// содержащий дочерний текстовый узел со значением 1.75
example.insertChildBefore( example.two[0], <part>1.75</part> );

/* На экран выводится:
<example>
  <one>"номер 1"</one>
  1.5
  <part>1.75</part>
  <two>2</two>
</example>
*/
trace( example );
```

В предыдущем примере кода был создан *смешанный элемент*. Смешанный элемент – это элемент, содержащий и вложенные текстовые узлы, и вложенные элементы. Элемент `example` смешанный, потому что содержит не только дочерние элементы (`<one>`, `<part>` и `<two>`), но и вложенный текстовый узел (со значением 1,5). Это вполне допустимо в XML и демонстрирует, что у элемента может быть много дочерних элементов, а не только один текстовый узел.

См. также

Рецепт 20.3.

20.5. Добавление атрибутов в элемент XML

Задача

Требуется добавить атрибуты в элемент XML.

Решение

Присваивание атрибутов узлам элементов в синтаксисе E4X осуществляется с помощью оператора `@`.

Обсуждение

Для присваивания атрибутов элементу в E4X предназначен оператор `@`; базовый синтаксис такой:

```
узелЭлемента.@имяАтрибута = "значение";
```

После переменной узла элемента ставится оператор «точка» (`.`), за которым следует оператор `@`. Сразу после символа `@` задается имя атрибута и простым выражением присваивания задается значение атрибута. Значение в правой части выражения присваивания перед присваиванием преобразуется в строку:

```
// Создаем экземпляр XML
var example:XML = <example><someElement/></example>;

// Добавляем атрибуты в узел элемента someElement
example.someElement.@number = 12.1;
example.someElement.@string = "пример";
example.someElement.@boolean = true;
example.someElement.@array = ["a", null, 7, undefined, "c"];

/* На экран выводится:
<example>
  <someElement number="12.1" string="пример" boolean="true"
  array="a,,7,,c"/>
</example>
*/
trace( example );
```

Этот синтаксис требует, чтобы имя атрибута было действительным именем переменной. Это означает, что имена атрибутов могут включать числа, буквы и символы подчеркивания, первый символ не может быть числом. Однако имена XML-атрибутов могут содержать и другие символы, которые при использовании с оператором `@` приводят к ошибке компиляции. Атрибуты, имена которых содержат недопустимые для имени переменной символы, создаются при помощи синтаксиса с квадратными скобками. Например:

```
example.someElement.@["bad-variable-name"] = "да";
```

Для динамического создания атрибутов можно применять и квадратные скобки, компонуя имя атрибута из значений переменных. Например:

```
example.someElement.@["color" + num] = "красный";
```

См. также

Рецепт 20.9.

20.6. Чтение элементов дерева XML

Задача

Требуется извлечь дочерние элементы объекта *XML*.

Решение

Обратитесь к методу *elements( )*, который возвратит все элементы в виде *XMLList*, и цикл *for each* – для перебора элементов списка.

Обсуждение

Часто возникает необходимость «пройти» (обойти) по дереву XML, чтобы извлечь или проверить один или более его элементов. Это удобный способ поиска заданного элемента или обработки элементов, порядок расположения которых неизвестен или неважен.

E4X предоставляет удобный метод *elements( )* (элементы), возвращающий все дочерние узлы элементов XML-объекта. Поместив этот метод в цикл *for each*, можно совершать обход дерева XML:

```
var menu:XML = <menu>
                 <menuitem label="Файл">
                   <menuitem label="Новый"/>
                 </menuitem>
                 <menuitem label="Помощь">
                   <menuitem label="О программе"/>
                 </menuitem>
                 Это текстовый узел
               </menu>;

for each ( var element:XML in menu.elements() ) {
  /* На экран выводится:
  Файл
  Помощь
  */
  trace( element.@label );
}
```

Данный пример показывает, что метод *elements( )* возвращает только дочерние узлы объекта XML типа *element* (игнорируя узлы другого типа, например текстовые). Он также возвращает только элементы, являющиеся прямыми потомками XML-объекта. Вызов *menu.elements( )* в предыдущем фрагменте кода не возвращает элементы меню `Новый` и `О программе`, потому что они не являются прямыми дочерними элементами `menu`. Чтобы обойти все дерево, необходимо создать рекурсивную функцию:

```
var menu:XML = <menu>
                 <menuitem label="Файл">
                   <menuitem label="Новый"/>
```

```
                </menuitem>
                <menuitem label="Помощь">
                  <menuitem label="О программе"/>
                </menuitem>
                Это текстовый узел
              </menu>;

/* На экран выводится:
Файл
Новый
Помощь
О программе
*/
walk( menu );

// Рекурсивная функция, выбирающая все элементы дерева XML
function walk( node:XML ):void {
  // Перебираем все дочерние элементы узла
  for each ( var element:XML in node.elements() ) {
    // Выводим атрибут label
    trace( element.@label );
    // Рекурсивно обходим дочерний элемент,
    // чтобы выбрать его дочерние элементы
    walk( element );
  }
}
```

См. также

Рецепты 20.7, 20.8 и 20.9.

20.7. Поиск элементов по имени

Задача

Требуется найти элемент по имени узла, а не по положению в иерархии XML.

Решение

Обратитесь к синтаксису E4X и посредством оператора «точка» переходите («dot down») к конкретному элементу.

Обсуждение

E4X позволяет работать с XML-объектами так же просто, как и с обычными объектами. Доступ к узлам элементов осуществляется так же, как к свойствам объекта; например:

```
var fruit:XML = <fruit><name>Яблоко</name></fruit>;

// На экран выводится: Яблоко
trace( fruit.name );
```

Как видите, чтобы получить доступ к элементу name (имя) пакета XML, используется оператор «точка» (.) и прямая ссылка на name XML-экземпляра fruit (фрукт). Для более глубоко вложенных элементов XML-дерева указывается последовательность точек и имен элементов:

```
var author:XML = <author><name><firstName>Деррон</firstName></name></author>;

// На экран выводится: Деррон
trace( author.name.firstName );
```

Краткая запись для организации доступа к глубоко вложенным узлам элементов, когда точный путь к узлу неизвестен, – двойной оператор «точка». Например, можно опустить путь и напрямую обратиться к узлу firstName (имя):

```
var author:XML = <author><name><firstName>Деррон</firstName></name></author>;

// На экран выводится: Деррон
trace( author..firstName );
```

Двойной оператор «точка» применим для любого уровня вложенности, хоть второго, хоть десятого.

Если существует несколько узлов элементов с одинаковыми именами, доступ к ним осуществляется посредством целочисленных индексов, которые указываются в квадратных скобках, как при работе с массивами. Например:

```
var items:XML = <items>
                  <item>
                    <name>Яблоко</name>
                    <color>красный</color>
                  </item>
                  <item>
                    <name>Апельсин</name>
                    <color>оранжевый</color>
                  </item>
                </items>;

// На экран выводится: Яблоко
trace( items.item[0].name );
// На экран выводится: Апельсин
trace( items.item[1].name );
```

items.item возвращает *XMLList* с двумя элементами, каждый из которых представляет узел элементов item XML-литерала. К первому узлу элементов item можно обратиться по индексу 0, ко второму – по индексу 1. Индекс первого элемента всегда равен 0, индекс последнего элемента всегда на единицу меньше, чем общее число элементов. Для подсчета числа выбранных элементов применяется метод *length( )*:

```
// На экран выводится: 2
trace( items.item.length() );
```

Если требуется проверить все узлы элементов с определенным именем и количество таких элементов не известно, используется цикл *for each*:

```
var items:XML = <items>
                  <item>
                    <name>Яблоко</name>
                    <color>красный</color>
                  </item>
                  <item>
                    <name>Апельсин</name>
                    <color>оранжевый</color>
                  </item>
                </items>;

// Используем синтаксис двойной точки,
// таким образом, можно опустить путь
for each ( var name:XML in items..name ) {
  /* На экран выводится:
  Яблоко
  Апельсин
  */
  trace( name );
}
```

Как и в других случаях применения оператора «точка», он может быть заменен квадратными скобками. Это полезно, когда в качестве имени искомого элемента выступает значение переменной:

```
var nodeName:String = "color";

var fruit:XML = <fruit><color>красный</color></fruit>;

// На экран выводится: красный
trace( fruit[nodeName] );
```

Квадратные скобки не могут использоваться с двойным оператором «точка». Например, следующий код приводит к ошибке:

```
trace( fruit..[nodeName] ); // Формируется ошибка компиляции
```

См. также

Рецепт 20.14.

20.8. Чтение текстовых узлов и их значений

Задача

Требуется извлечь значение текстового узла.

Решение

Используйте синтаксис E4X или, в качестве альтернативы, метод *text( )* для возвращения объекта *XMLList*, содержащего текстовые уз-

лы элемента. Затем с помощью метода *toString( )* преобразуйте значение текстового узла в строку. Обратитесь к функциям преобразования, например *int( )* или *Number( )*, для преобразования значения в другой тип данных или предоставьте Flash Player возможность автоматически преобразовать значение в определенный тип.

Обсуждение

Рецепт 20.4 рассматривает текстовые узлы и их создание в объектах XML. Данный рецепт объясняет, как извлекать значение текстового узла. Сначала необходимо получить ссылку на текстовый узел, во многом аналогично узлам элементов, что обсуждалось в рецептах 20.6 и 20.7. Имея ссылку на текстовый узел, можно получить его значение, вызывая метод *toString( )* или другой метод преобразования. Рассмотрим следующий пакет XML:

```
<book>
  <title>ActionScript 3.0. Сборник рецептов</title>
</book>
```

Корневым узлом данного XML-пакета является `<book>` (книга), содержащий узел `<title>` как дочерний элемент. Элемент `<title>`, в свою очередь, включает вложенный узел: текстовый узел, имеющий значение `ActionScript 3.0. Сборник рецептов`.

Если этот XML-пакет присвоен объекту *XML*, можно использовать синтаксис E4X и оператор «точка» для доступа к узлу элемента `title` по имени, а также метод *toString( )* для доступа к его дочернему текстовому узлу:

```
var book:XML = <book>
                 <title>ActionScript 3.0. Сборник рецептов</title>
               </book>;

// Используем синтаксис E4X для организации доступа
// к элементу title и присваиваем его значение переменной.
var title:String = book.title.toString();

// На экран выводится: ActionScript 3.0. Сборник рецептов
trace( title );
```

В этом примере преобразование узла элемента `title` в тип *String* осуществляется явно вызовом метода *toString( )*. На самом деле вызов *toString( )* можно опустить, потому что Flash Player проводит преобразование автоматически. Но явный вызов *toString( )* хорош тем, что делает код более прозрачным и понятным.

Аналогично можно преобразовать текстовый узел в другой тип данных. В следующем примере демонстрируется преобразование текстовых узлов в различные типы с помощью функций преобразования *Boolean( )* и *int( )* и автоматически осуществляемое Flash Player преобразование в тип *Number* (без явного вызова метода *Number( )*):

```
var example:XML = <example>
                    <bool>true</bool>
                    <integer>12</integer>
                    <number>.9</number>
                  </example>;

// Преобразуем текстовый узел "true" в Булево true
var bool:Boolean = Boolean( example.bool );

// Преобразуем текстовый узел "12" в целое
var integer:int = int( example.integer );

// Преобразуем текстовый узел ".9" в число
var number:Number = example.number;

/* На экран выводится:
true
12
0.9
*/
trace( bool );
trace( integer );
trace( number );
```

Этот пример немного неточен. Функция преобразования *Boolean( )* может не обеспечить того результата, который предполагается. Изменение значения текстового узла в элементе `<bool>` с `true` на `false` приводит к тому, что переменной `bool` будет задано значение `true`, а не `false`, как ожидается. Такое поведение заложено при проектировании функции, хотя и кажется немного странным. Простой прием для преобразования строки `false` в Булево `false` – вызвать метод *toLowerCase( )* и сравнить возвращаемое значение со строкой `true`:

```
var bool:Boolean = example.bool.toLowerCase() == "true";
```

До сих пор мы работали с узлами элементов, содержащими всего один дочерний текстовый узел. А что же смешанные элементы? При преобразовании смешанного элемента в строку получается форматированная строка XML:

```
var fruit:XML = <fruit>
                  <name>Яблоко</name>
                  По яблоку в день...
                </fruit>;

// Здесь необходимо явно вызвать toString()
var value:String = fruit.toString();

/* На экран выводится:
<fruit>
  <name>Яблоко</name>
  По яблоку в день...
</fruit>
*/
trace( value );
```

В таких ситуациях возвращение только текстовых узлов элемента обеспечит метод *text( )*. Как показано в рецепте 20.6, для перебора всех узлов организуется цикл *for each*:

```
var fruit:XML = <fruit>
                  <name>Яблоко</name>
                  По яблоку в день...
                </fruit>;

for each ( var textNode:XML in fruit.text() ) {
  // На экран выводится: По яблоку в день...
  trace( textNode );
}
```

См. также

Рецепты 20.4, 20.6 и 20.7.

20.9. Чтение атрибутов элемента

Задача

Требуется извлечь атрибуты элемента.

Решение

Список атрибутов элемента возвращает метод *attributes( )*. В качестве альтернативы может выступать оператор @ синтаксиса E4X или метод *attribute( )* для организации доступа к атрибуту по имени.

Обсуждение

Список атрибутов определенного узла элементов в виде объекта *XMLList* возвращается методом *attributes( )*. Возвращенный *XMLList* является индексируемым, как экземпляр *Array*. По индексу можно получить значение атрибута:

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

// Используем метод attributes() и сохраняем результаты как XMLList
var attributes:XMLList = fruit.attributes();

// На экран выводится: Яблоко
trace( attributes[0] );
// На экран выводится: красный
trace( attributes[1] );
```

В данном примере извлекаются только значения атрибутов. Имя атрибута позволяет получить метод *name( )*. В следующем примере проверяется имя второго атрибута (расположенного под индексом 1):

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

// На экран выводится: color
trace( fruit.attributes()[1].name() );
```

Чтобы выбрать все атрибуты элемента, используется цикл *for each*. Чтобы получить имя и значение атрибута, применяется метод *name( )* в сочетании со ссылкой на атрибут. В следующем примере метод *toString( )* вызывается для ссылки на атрибут явно, чтобы преобразовать значение атрибута в строку, как в рецепте 20.8:

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

for each ( var attribute:XML in fruit.attributes() ) {
  /* На экран выводится:
  name = Яблоко
  color = красный
  */
  trace( attribute.name() + " = " + attribute.toString() );
}
```

Если необходимо получить конкретный атрибут и известно его имя, не составит никакого труда сделать это с помощью синтаксиса E4X. Для этого посредством оператора «точка» необходимо обратиться к конкретному узлу элемента и затем с помощью оператора `@` указать имя атрибута, чтобы получить его значение:

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

// На экран выводится: красный
trace( fruit.@color );
```

Кроме того, можно вызвать метод *attribute( )*, передав в него имя атрибута:

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

// На экран выводится: красный
trace( fruit.attribute("color") );
```

Также, чтобы организовать доступ ко всем атрибутам элемента, аналогично способу с методом *attributes( )*, можно прибегнуть к оператору `@` с групповым символом (`*`):

```
var fruit:XML = <fruit name="Яблоко" color="красный" />;

// На экран выводится: Яблоко
trace( fruit.@*[0] );

// На экран выводится: красный
trace( fruit.@*[1] );

// На экран выводится: 2
trace( fruit.@*.length() );
```

Атрибуты всегда возвращаются как *XMLList*, поэтому они проиндексированы, что упрощает доступ к ним.

Синтаксис E4X – исключительно мощное средство. Рассмотрим пакет XML, содержащий ряд элементов, причем некоторые из них имеют атрибут `price` (цена). Следующий фрагмент кода показывает, как можно

использовать E4X для вычисления суммарной стоимости элементов, имеющих атрибут `price`:

```
// Создаем воображаемую корзину покупок
var cart:XML = <cart>
                 <item price=".98">мелки</item>
                 <item price="3.29">карандаши</item>
                 <group>
                   <item price=".48">синяя ручка</item>
                   <item price=".48">черная ручка</item>
                 </group>
               </cart>;

// Создаем переменную total для представления
// общей стоимости товаров в корзине
var total:Number = 0;

// Выбираем каждый атрибут price и добавляем
// его значение к промежуточной сумме
for each ( var price:XML in cart..@price ) {
  total += parseFloat(price);
}

// На экран выводится: 5.23
trace( total );
```

Двойной оператор «точка» перед оператором `@` в синтаксисе E4X обеспечивает поиск атрибутов `price` по всему XML-дереву узла элементов `cart`.

Как показано в рецепте 20.5, для динамического доступа к атрибуту по имени либо в случае, если имя атрибута содержит недопустимые для имен переменных символы, можно использовать синтаксис квадратных скобок:

```
var example:XML = <example bad-variable-name="да" color12="синий" />;
var num:Number = 12;

// На экран выводится: да
trace( example.@["bad-variable-name"] );

// На экран выводится: синий
trace( example.@["color" + num] );
```

См. также

Рецепты 20.5 и 20.8.

20.10. Удаление элементов, текстовых узлов и атрибутов

Задача

Требуется удалить из объекта *XML* узел элементов, текстовый узел или атрибут.

Решение

Это делается при помощи ключевого слова `delete`.

Обсуждение

В предыдущих рецептах мы научились добавлять узлы элементов, текстовые узлы и атрибуты в объекты *XML*. Также было показано, как организовывать работу с ними (чтение и перебор). Но что если требуется удалить определенный элемент или атрибут? Достаточно указать ключевое слово `delete`, а за ним – элемент (атрибут), подлежащий удалению:

```
var example:XML = <example>
                    <fruit color="Красный">Яблоко</fruit>
                    <vegetable color="Зеленый">Брокколи</vegetable>
                    <dairy color="Белый">Молоко</dairy>
                  </example>;

// Удаляем атрибут color из элемента fruit
delete example.fruit.@color;

// Полностью удаляем элемент dairy
delete example.dairy;

// Удаляем текстовый узел из узла элементов vegetable
delete example.vegetable.text()[0];

/* На экран выводится:
<example>
  <fruit>Яблоко</fruit>
  <vegetable color="Зеленый"/>
</example>
*/
trace( example );
```

Особый интерес в предыдущем примере вызывает удаление текстового узла. В случае применения определенных методов объекта *XML*, таких как *text( )* и *elements( )*, или, в определенных ситуациях, синтаксиса E4X возвращается экземпляр *XMLList* с множеством элементов. Ключевое слово `delete` применимо только к одному элементу. Таким образом, чтобы указать конкретный элемент *XMLList*, который следует удалить, необходимо использовать запись с квадратными скобками. Чтобы удалить все элементы *XMLList*, организуется цикл *for* и перебор элементов в обратном порядке:

```
var example:XML = <example>
                    <fruit color="красный" name="Яблоко" />
                  </example>;

// Получаем XMLList атрибутов для fruit
var attributes:XMLList = example.fruit.@*;

// Перебираем элементы в обратной последовательности,
```

```
// чтобы удалить все атрибуты. Удаляя элементы с конца
// массива, мы избегаем проблемы с изменением индексов.
for ( var i:int = attributes.length() - 1; i >= 0; i-- ) {
    delete attributes[i];
}

/* На экран выводится:
<example>
  <fruit/>
</example>
*/
trace( example );
```

20.11. Загрузка XML

Задача

Требуется загрузить данные XML из документа XML или серверного сценария, который генерирует XML.

Решение

Метод *URLLoader.load( )*, свойству `dataFormat` которого задано значение `URLLoaderDataFormat.TEXT`, обеспечит загрузку данных в виде простого текста. Используйте обработчик события `complete` и преобразуйте текст в экземпляр *XML*.

Обсуждение

В предыдущих версиях ActionScript загрузка XML-файла осуществлялась посредством вызова метода *load( )* непосредственно для объекта *XML*. В ActionScript 3.0 отправка и загрузка данных была выделена в класс *URLLoader* и родственные ему классы. В ActionScript 3.0 нет специальных средств загрузки XML, что предоставляет свободу действий разработчикам для реализации собственных решений.

Процесс загрузки файла XML хотя и многошаговый, но относительно безболезненный. Сначала необходимо создать экземпляр *URLLoader* для загрузки данных с URL. Чтобы *URLLoader* загружал данные как простой текст, его свойству `dataFormat` должно быть задано значение `URLLoaderDataFormat.Text`. Добавляем слушателя события `complete`, чтобы получать уведомление о завершении загрузки данных. В обработчике события `complete` для преобразования загруженных данных в объект *XML* задействуется одна из техник, описанных в рецепте 20.2. И наконец, для запуска процесса загрузки вызывается метод *URLLoader.load( )*, в который передается экземпляр *URLRequest*, указывающий на URL файла XML. Законченный пример выглядит так:

```
package {
  import flash.display.*;
  import flash.events.*;
```

```
import flash.net.*;
import flash.utils.*;

public class LoadXMLExample extends Sprite {

  public function LoadXMLExample() {
    var loader:URLLoader = new URLLoader();
    loader.dataFormat = URLLoaderDataFormat.TEXT;
    loader.addEventListener( Event.COMPLETE, handleComplete );
    loader.load( new URLRequest( "example.xml" ) );
  }

  private function handleComplete( event:Event ):void {
    try {
      // Преобразуем загруженный текст в экземпляр XML
      var example:XML = new XML( event.target.data );
      // Здесь example готов к использованию с E4X
      trace( example );

    } catch ( e:TypeError ) {
      // Если мы попали сюда, значит, не получилось преобразовать
      // загруженный текст в экземпляр XML, вероятно, потому что
      // он отформатирован несоответствующим образом.
      trace( "Не получилось преобразовать текст в XML" );
      trace( e.message );
    }
  }
}
```

В предыдущем примере обратите внимание на использование блока *try...catch* в методе `handleComplete`. Если XML-файл не содержит действительной XML-разметки, при попытке преобразования загруженных данных в экземпляр *XML* формируется *TypeError* (ошибка приведения типов данных). Ошибка перехватывается блоком *catch*, что позволяет красиво обработать сбой при синтаксическом разборе.

Более подробно об работе с *URLLoader* рассказано в главе 19.

См. также

Рецепт 20.2 и главу 19, в частности рецепт 19.3.

20.12. Загрузка XML из других доменов

Задача

Требуется загружать XML из других доменов, а не только из того, в котором находится файл *.swf*.

Решение

Сделайте в *файле политики crossdomain.xml* удаленного домена запись, разрешающую доступ к нему из домена файла *.swf*.

Обсуждение

Об использовании *файла политики crossdomain.xml* рассказано в рецепте 3.12.

См. также

Рецепт 3.12.

20.13. Отправка XML

Задача

Требуется отправить данные XML в сценарий, выполняющийся на стороне сервера.

Решение

Создайте экземпляр *URLRequest*, содержащий данные XML, подлежащие отправке. Обратитесь к методу *flash.net.sendToURL( )*, если хотите отправить данные и проигнорировать ответ сервера. Метод *flash.net.navigateToURL( )* позволит отправить данные и открыть ответ сервера в определенном окне броузера, а метод *URLLoader.load( )* – отправить данные и загрузить ответ в файл *.swf*.

Обсуждение

XML используется для передачи данных в приложения и из них, в данном случае – роликов Flash. Поэтому было бы странно, если бы объекты XML создавались в ролике Flash для использования только в нем. Обычно данные XML загружаются из другого источника, во Flash создаются данные XML для отправки в другое приложение или и то, и другое.

В данном рецепте рассматривается отправка данных XML из Flash в другое приложение, что может потребоваться во многих случаях. Например, во Flash-игре XML может задействоваться для передачи на сервер имени пользователя и счета. Иногда пакеты XML посылаются на сервер для активизации какой-либо функциональности сервера. Такой процесс иногда называют *удаленным вызовом процедур* (Remote Procedure Call, RPC), и здесь для передачи на сервер информации вызова функции (имени функции, параметров и т. п.) может применяться XML. Существует формальная спецификация использования XML в таких случаях – XML-RPC (см. *http://www.xmlrpc.com*). Так что, как видите, причины отправки XML на сервер весьма разнообразны.

Как обсуждалось в рецепте 20.11 и главе 19, ActionScript 3.0 объединил отправку и загрузку данных в методах пакета *flash.net*. Ранее класс *XML* содержал и метод *send( )*, и метод *sendAndLoad( )* для отправки XML на сервер, но в ActionScript 3.0 место этих методов занял

экземпляр *URLRequest*. В рецептах 19.6 и 19.7 рассматриваются основные приемы использования экземпляра *URLRequest* для отправки и получения данных.

Рассмотрим законченный рабочий пример. Ниже создается необходимый клиентский код на ActionScript, после чего требуется выбрать одно из серверных решений. Выбирайте то решение, которое поддерживается вашим сервером (или персональным компьютером, если он выступает в качестве сервера). Сценарии, выполняющиеся на стороне сервера, могут быть написаны на Perl, PHP и ColdFusion.

Прежде всего создается новый класс ActionScript 3.0:

```
package {
  import flash.display.*;
  import flash.text.*;
  import flash.filters.*;
  import flash.events.*;
  import flash.net.*;

  public class XMLSendLoadExample extends Sprite {

    private var _message:TextField;
    private var _username:TextField;
    private var _save:SimpleButton;

    public function XMLSendLoadExample() {
      initializeDisplay();
    }

    private function initializeDisplay():void {
      _message = new TextField();
      _message.autoSize = TextFieldAutoSize.LEFT;
      _message.x = 10;
      _message.y = 10;
      _message.text = "Enter a user name";

      _username = new TextField();
      _username.width = 100;
      _username.height = 18;
      _username.x = 10;
      _username.y = 30;
      _username.type = TextFieldType.INPUT;
      _username.border = true;
      _username.background = true;

      _save = new SimpleButton();
      _save.upState = createSaveButtonState( 0xFFCC33 );
      _save.overState = createSaveButtonState( 0xFFFFFF );
      _save.downState = createSaveButtonState( 0xCCCCCC );
      _save.hitTestState = _save.upState;
      _save.x = 10;
      _save.y = 50;
      // По щелчку кнопки save вызывается метод handleSave
```

```
    _save.addEventListener( MouseEvent.CLICK, handleSave );

    addChild( _message );
    addChild( _username );
    addChild( _save );
  }

  // Создаем состояние кнопки с определенным цветом фона
  private function createSaveButtonState( color:uint ):Sprite {
    var state:Sprite = new Sprite();

    var label:TextField = new TextField();
    label.text = "Сохранить";
    label.x = 2;
    label.height = 18;
    label.width = 30;
    var background:Shape = new Shape();
    background.graphics.beginFill( color );
    background.graphics.lineStyle( 1, 0x000000 );
    background.graphics.drawRoundRect( 0, 0, 32, 18, 9 );
    background.filters = [ new DropShadowFilter( 1 ) ];

    state.addChild( background );
    state.addChild( label );
    return state;
  }

  private function handleSave( event:MouseEvent ):void {
    // Формируем случайный счет, сохраняемый с username
    var score:int = Math.floor( Math.random() * 10 );

    // Создаем новый экземпляр XML, содержащий данные,
    // которые должны быть сохранены
    var dataToSave:XML = <gamescore>
                           <username>{_username.text}</username>
                           <score>{score}</score>
                         </gamescore>;

    // Направляем запрос в сценарий, который будет обрабатывать XML
    var request:URLRequest = new URLRequest( "/gamescores.cfm" );
    // Присваиваем свойству data экземпляр XML dataToSave,
    // чтобы отправить данные XML на сервер
    request.data = dataToSave;
    // Задаем contentType, чтобы показать, что передаются данные XML
    request.contentType = "text/xml";
    // Используем метод post для отправки данных
    request.method = URLRequestMethod.POST;

    // Создаем URLLoader для обработки отправки и загрузки данных XML
    var loader:URLLoader = new URLLoader();
    // По завершении загрузки ответа сервера вызываем handleResponse
    loader.addEventListener( Event.COMPLETE, handleResponse );
    // Наконец, отправляем данные XML по заданному URL
    loader.load( request );
```

```
    }

    private function handleResponse( event:Event ):void {
      try {
        // Делаем попытку преобразовать ответ сервера в XML
        var success:XML = new XML( event.target.data );

        // Проверяем значение узла элементов success
        if ( success.toString() == "1" ) {
          _message.text = "Сохранено успешно.";
        } else {
          _message.text = "В процессе сохранения произошла ошибка.";
        }

      } catch ( e:TypeError ) {
        // Выводим сообщение об ошибке, поскольку ответ сервера не понят
        _message.text = "Не удалось разобрать XML-ответ сервера.";
      }
    }
  }
}
```

Свойство `contentType` (тип содержимого) экземпляра *URLRequest* по умолчанию имеет значение `application/x-www-form-urlencoded`, поэтому важно при отправке данных XML присвоить ему значение `text/xml`. Кроме того, чтобы отправлять данные на сервер посредством HTTP-метода `POST`, свойству `method` запроса необходимо присвоить значение *URLRequestMethod.POST*.

Конечно, в реальном примере счет генерировался бы в зависимости от результатов игры. В данном случае требовалось продемонстрировать отправку и прием XML, поэтому счет генерировался случайным образом.

Следующий шаг – создать сценарий, выполняющийся на стороне сервера. Первым приводим вариант на Perl. Если вы будете использовать его, поместите следующий код в текстовый файл *gamescores.cgi* (или *gamescores.pl*) в каталог вашего веб-сервера, для которого активирован доступ к CGI (обычно *cgi* или *cgi-bin*).

```
#!/usr/bin/perl

# Flash/Perl+CGI XML interaction demo
# Arun Bhalla (arun@groogroo.com)

use strict;
use XML::Simple;
use CGI;

my $ScoreFile = "scores.txt";

# Здесь мы предполагаем, что данный сценарий CGI
# получает XML в форме text/xml посредством POST.
# Поэтому XML предоставляется в сценарий через STDIN.
```

```
my $input = XMLin(join('',<STDIN>));

# Отправляем HTTP-заголовок
print CGI::header('text/xml');

# Пытаемся открыть файл счета для записи или возвращаем сообщение об ошибке.
open(SCORES, ">> $ScoreFile") || (printMessage(0) &&
                 die "Ошибка при открытии $ScoreFile");

# Сохраняем счет в текстовом файле с разделяющими символами "|".
print SCORES join('|', $input->{username}, $input->{score}), "\n";

# Возвращаем результат в виде XML.
printMessage(1);

# Подпрограмма для вывода результата в виде XML.
sub printMessage {
  my $value = shift;
  my $message = {};
  $message->{success} = $value;
  print XMLout($message, keeproot => 1, rootname => 'success');
}
```

Следующий блок кода представляет пример сценария на ColdFusion. Поместите данный код в ColdFusion-страницу *gamescores.cfm* в каталог веб-сервера, который поддерживает данный тип приложений:

```
<cfsilent>
<cfsetting enablecfoutputonly="Yes">
<cfset success = 0>
<cftry>
  <!--- пакет XML, посланный Flash. --->
  <cfset scores_xml = XmlParse( getHTTPRequestData(  ).content ) >

  <!--- Синтаксический разбор пакета XML, присланного Flash. --->

  <!--- Из документа XML выбирается username и score и сохраняются как
        локальные переменные, чтобы упростить работу с ними. // --->
  <cfset username = scores_xml.gamescore.username.XmlText >
  <cfset score = scores_xml.gamescore.score.XmlText >

  <!--- Последний счет добавляется в файл счетов. Он также
        может быть сохранен в базе данных или другом документе XML. // --->
  <cffile action="APPEND" file="#ExpandPath( 'scores.txt' )#"
  output="#username#|#score#|#getHTTPRequestData(  ).content#" addnewline="
Yes">
  <cfset success = 1 >
  <cfcatch type="Any">
    <cfset success = 0 >
    <cffile action="APPEND" file="#ExpandPath( 'attempts.txt' )#
                                                          " output="ERROR"
    addnewline="Yes">
  </cfcatch>
</cftry>
</cfsilent>
<cfcontent type="text/xml">
```

```
<cfoutput><?xml version="1.0" ?><success>#success#</success></cfoutput>
<cfsetting showdebugoutput="no" enablecfoutputonly="No">
```

Если на вашем сервере используется PHP, поместите следующий код в PHP-страницу *gamescores.php* в каталог веб-сервера, который сможет обработать запрос к данной странице:

```
<?php

// Читаем XML из Raw Post Data.
$xml = $GLOBALS['HTTP_RAW_POST_DATA'];

// Обрабатываем XML, используя расширение DOM PHP.
$document = xmldoc($xml);

// Читаем корневой элемент <gameinfo>.
$rootElement = $document->root(  );

// Читаем дочерние узлы <username> и <score>.
$childNodes = $rootElement->children(  );

$data = "";

// Перебираем дочерние узлы и помещаем их в массив.
foreach($childNodes as $childNode){
  // Добавляем данные в массив;
  $name = $childNode->tagName(  );
  $value = $childNode->get_content(  );
  $data[$name] = $value;
}

// Добавляем данные в scores.txt ( формат: username|score)
$fp = fopen("scores.txt","a+");
$dataString = $data['username'] . "|" . $data['score'] . "\n";
fputs($fp,$dataString,strlen($dataString));
fclose($fp);

// Возвращаем код успешности операции во Flash
echo "<success>1</success>";
?>
```

У всех трех сценариев есть общая черта: они возвращают XML-ответ с корневым узлом `<success>`, содержащий текстовый узел со значением 1 для обозначения успешности операции сохранения и 0, который является признаком ошибки.

Создав ролик Flash и серверный сценарий, чтобы протестировать их, достаточно запустить ролик и нажать кнопку. Ролик должен получить ответ об успешности операции, а в каталоге сервера, в котором находится сценарий, появится файл *scores.txt*, содержащий данные, предоставленные роликом Flash.

См. также

Рецепты 19.6, 19.7 и 20.11.

20.14. Поиск в XML-документах

Задача

Требуется найти в объекте *XML* узлы или атрибуты, отвечающие определенным критериям.

Решение

Для выбора определенных значений из дерева XML примените к объекту *XML* синтаксис E4X вместе с *условием фильтрации*.

Обсуждение

В данной главе рассмотрен вопрос о том, как синтаксис E4X с объектами *XML* упрощает чтение и запись значений XML-дерева. Кроме того, что E4X прост в применении, он обладает исключительной мощью. Синтаксис E4X аналогичен использованию *XPath* для поиска в документах XML. У тех, кто хорошо знаком с основами *XPath*, работа с расширенными возможностями E4X (такими как фильтрация по условию) не должна вызвать затруднений. Фильтрация по условию позволяет выбирать узлы элементов, отвечающие условию определенного Булева выражения, используя синтаксис `.( условие )`, как будет показано ниже в данном рецепте.

Начнем с создания *XML* из литерала XML:

```
var foodgroup:XML = <foodgroup>
                      <fruits>
                        <fruit color="красный">Яблоко</fruit>
                        <fruit color="оранжевый">Апельсин</fruit>
                        <fruit color="зеленый">Груша</fruit>
                        <fruit color="красный">Арбуз</fruit>
                        <servings>3</servings>
                      </fruits>
                      <vegetables>
                        <vegetable color="красный">Помидор</vegetable>
                        <vegetable color="brown">Картошка</vegetable>
                        <vegetable color="зеленый">Брокколи</vegetable>
                        <servings>2</servings>
                      </vegetables>
                    </foodgroup>;
```

Если имена узлов элементов известны, обратиться к ним можно с помощью оператора «точка». Например, список всех узлов элементов `<fruit>` можно возвратить посредством следующего выражения E4X:

```
var fruitList:XMLList = foodgroup.fruits.fruit;
```

Если требуется какой-то определенный узел `<fruit>`, доступ к нему можно организовать, задавая в квадратных скобках значение его индекса:

```
var theApple:XML = foodgroup.fruits.fruit[0];
```

Если полный путь от корневого узла к искомому узлу (или узлам) неизвестен (или не имеет значения), применяется двойной оператор «точка», выбирающий все соответствующие узлы на любом уровне дерева XML. Приведенная ниже строка кода возвращает все узлы <vegetable> независимо от их положения в иерархии:

```
var vegetableList:XMLList = foodgroup..vegetable;
```

Звездочка (*) – это групповой символ для выбора «любого узла». Следующее выражение E4X присваивает переменной *servings* (порции) значение экземпляра *XMLList*, содержащего все узлы элементов <servings>, являющиеся дочерними элементами любых узлов, которые, в свою очередь, представляют собой дочерние элементы узла <foodgroup> (группа продуктов):

```
var servings:XMLList = foodgroup.*.servings;
```

Знак @ служит для обозначения атрибута. Следующий пример формирует *XMLList*, содержащий значения атрибутов color узлов <fruit>:

```
var colorValues:XMLList = foodgroup.fruits.fruit.@color;
```

Теперь рассмотрим фильтрацию по условию. Фильтрация по условию посредством синтаксиса .(*условие*) выбирает узлы элементов, отвечающие заданному условию. Условие определяется Булевым выражением. Фильтрации подвергается объект *XML* или *XMLList*, предшествующий выражению фильтра.

Допустим, что требуется выбрать все узлы <fruit>, где атрибут color имеет значение красный. Для этого сначала с помощью синтаксиса E4X и оператора «точка» формируется экземпляр *XMLList*, включающий все узлы элементов <fruit>, после чего проводится фильтрация посредством выражения @color == "красный":

```
/* На экран выводится:
<fruit color="красный">Яблоко</fruit>
<fruit color="красный">Арбуз</fruit>
*/
trace( foodgroup..fruit.( @color == "красный" ) );
```

Здесь в выражении, переданном в trace, происходят две вещи:

- Часть foodgroups..fruit возвращает *XMLList* всех узлов элементов <fruit> дерева XML.
- К *XMLList* узлов <fruit> применяется фильтрация по условию и создается новый *XMLList*, содержащий только те элементы <fruit>, которые соответствуют выражению фильтра, в данном случае это элементы, атрибуты color которых имеют значение красный. Как видите, оба узла элементов <fruit>, атрибут color которых равен красный, присутствуют в выводе trace.

Приведенный выше код выбирает узлы, представляющие красные фрукты. А что если требуется выбрать любой узел, атрибут color которого имеет значение красный? На помощь приходит звездочка, обеспечивающая выбор любого узла, в сочетании с фильтром, который проверяет наличие атрибута color, и если color существует, проверяет его значение:

```
/* На экран выводится:
<fruit color="красный">Яблоко</fruit>
<fruit color="красный">Арбуз</fruit>
<vegetable color="красный">Помидор</vegetable>
*/
trace( foodgroup..*.( hasOwnProperty( "@color" ) && @color == "красный" )  );
```

Булево выражение, выступающее в качестве условия, может быть любым выражением, возвращающим Булевы значения true или false. В предыдущем примере метод hasOwnProperty (имеет свойство) проверяет элемент на наличие атрибута color и, если находит его, проверяет его значение. Элемент добавляется в *XMLList*, возвращаемый выражением E4X, только если условие выполняется.

До сих пор фильтрация по условию применялась только к атрибутам. Однако она же позволяет выбирать узел элементов, имеющий текстовый узел с определенным значением. Это особенно полезно, если имеется документ XML с повторяющимися узлами элементов, содержащими дочерние узлы. Например, вот как можно вывести на экран значение атрибута color любого узла элементов <fruit>, узел элемента <name> которого содержит текстовый узел Яблоко:

```
var fruits:XML = <fruits>
                   <fruit color="красный">
                     <name>Яблоко</name>
                   </fruit>
                   <fruit color="оранжевый">
                     <name>Апельсин</name>
                   </fruit>
                   <fruit color="зеленый">
                     <name>Груша</name>
                   </fruit>
                   <fruit color="красный">
                     <name>Арбуз</name>
                   </fruit>
                 </fruits>;

// На экран выводится: красный
trace( fruits.fruit.(name == "Яблоко").@color );
```

Как видите, фильтрация по условию – это довольно мощное средство, особенно в сочетании с регулярными выражениями. В следующем примере с помощью регулярного выражения выбираются все узлы элементов <fruit>, имеющие дочерний узел <name>, в которых содержится текстовый узел, начинающийся с гласной:

```
var fruits:XML = <fruits>
                   <fruit color="красный">
                     <name>Яблоко</name>
                   </fruit>
                   <fruit color="оранжевый">
                     <name>Апельсин</name>
                   </fruit>
                   <fruit color="зеленый">
                     <name>Груша</name>
                   </fruit>
                   <fruit color="красный">
                     <name>Арбуз</name>
                   </fruit>
                 </fruits>;
/* На экран выводится:
<fruit color="красный">
  <name>Яблоко</name>
</fruit>
<fruit color="оранжевый">
  <name>Апельсин</name>
</fruit>
*/
trace( fruits.fruit.( /^[аеиоуэюяАЕИОУЭЮЯ].*/.test( name ) ) );
```

В предыдущем фрагменте кода создается регулярное выражение (между / и /), которое можно прочитать так: «начинается с гласной, в верхнем или нижнем регистре, за которой следует любой символ, повторяющийся любое количество раз». Для регулярного выражения вызывается метод *test( )*, чтобы сравнить его с передаваемым параметром (в данном случае, узлом элемента `<name>`, преобразованным в значение его текстового узла для рассматриваемого элемента `<fruit>`). Поскольку метод *test( )* возвращает Булево значение, его можно использовать как часть Булева условия фильтра.

Регулярные выражения подробно рассмотрены в главе 13.

См. также

Рецепты 20.7, 20.8, 20.9 и главу 13

20.15. Использование HTML и специальных символов в XML

Задача

Требуется использовать HTML или другой текст, содержащий символы, имеющие специальное значение в контексте XML.

Решение

Заключите текст в тег `CDATA`.

Обсуждение

Хотя в данном рецепте речь не идет об ActionScript как таковом, он имеет отношение к делу, потому что затрагивает вопросы, с которыми разработчики наверняка столкнутся при работе с XML во Flash или в других средствах разработки. В XML определенные символы имеют специальное значение. Например, знаки «больше чем» и «меньше чем» обозначают границы тегов XML. Любой из этих символов в тексте XML-документа породит ошибки при его синтаксическом разборе. Например:

```
<example>a < b</example>
```

Как видите, здесь текстовый узел содержит текст `a < b`. Но поскольку знак «меньше чем» имеет специальное значение в контексте документа XML, данное выражение приводит к ошибке синтаксического анализатора. Другой распространенный пример – размещение HTML в XML-документе; например:

```
<htmlExample><a href="http://www.darronschall.com">Деррон</a></htmlExample>
```

В этом случае HTML трактуется как XML, а не как строка. В предыдущий пакет XML входят корневой узел `<htmlExample>` с дочерним узлом `<a>`, содержащим текстовый узел со строковым значением `Деррон`.

В подобном случае текст можно заключить в тег `CDATA` (Character Data – символьные данные), который не обрабатывается анализатором XML. Он просто воспринимает эти данные как строку.

Тег `CDATA` начинается с `<![CDATA[` и заканчивается `]]>`. Следовательно, первый пример можно изменить так:

```
<example><![CDATA[a < b]]></example>
```

Второй пример можно переписать следующим образом:

```
<htmlExample><![CDATA[<a href="http://www.darronschall.com">Деррон</a>]]></
htmlExample>
```

При синтаксическом разборе XML символьные данные, заключенные в тег `CDATA`, трактуются как одна строка, а сам тег `CDATA` не рассматривается. Таким образом, в первом примере получаем текстовый узел `a < b`, а не `<![CDATA[a < b]]>`. Во втором примере пакет XML интерпретируется как корневой узел `<htmlExample>` с одним текстовым узлом, строковым значением которого является вся HTML-ссылка. Из-за тега `CDATA` `<a>` рассматривается не как дочерний узел, как это было раньше, а как часть строкового значения текстового узла.