

ЮРИЙ ШУТЕНКО



Visual FoxPro

ДЛЯ ПРОФЕССИОНАЛОВ



Visual FoxPro +
Web-ТЕХНОЛОГИИ:
ASP.NET, LINQ, Silverlight,
JavaScript, AJAX, JSON

Visual FoxPro +
ТЕХНОЛОГИИ WINDOWS:
COM, DCOM и COM+,
WINDOWS Shell, ActiveX

ОБМЕН ДАННЫМИ
С ДРУГИМИ
ПРИЛОЖЕНИЯМИ

РАСШИРЕНИЕ
ВОЗМОЖНОСТЕЙ:
НАСТРОЙКА ИНТЕРФЕЙСА,
ТРЮКИ И СЕКРЕТЫ

PRO
ПРОФЕССИОНАЛЬНОЕ
ПРОГРАММИРОВАНИЕ

+ CD

Юрий Шутенко

Visual FoxPro

ДЛЯ ПРОФЕССИОНАЛОВ

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.068
ББК 32.973.26-018.1
Ш97

Шутенко Ю. Т.

Ш97 Visual FoxPro для профессионалов. — СПб.: БХВ-Петербург, 2009. — 576 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0307-5

Книга посвящена расширению возможностей приложений Visual FoxPro за счет использования современных технологий. Показано применение различных Web-технологий, таких как ASP.NET, LINQ, Silverlight, JavaScript, AJAX, JSON и др. Описаны способы размещения и получения данных в Интернете. Рассмотрено применение Windows-технологий: COM, DCOM и COM+, Windows Shell, ActiveX и др. Показана организация обмена данными с различными СУБД (MySQL, SQL Server) и другими приложениями. Уделено внимание вопросам расширения возможностей VFP за счет настроек интерфейса и применения различных трюков при программировании. Компакт-диск содержит исходные тексты программ, классов и демонстрационных примеров, описанных в книге.

Для разработчиков

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зам. главного редактора	<i>Игорь Шишигин</i>
Зав. редакцией	<i>Григорий Добин</i>
Редактор	<i>Екатерина Капалыгина</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Корректор	<i>Зинаида Дмитриева</i>
Дизайн серии	<i>Инны Тачиной</i>
Оформление обложки	<i>Елены Беляевой</i>
Зав. производством	<i>Николай Тверских</i>

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 29.10.08.

Формат 70×100^{1/16}. Печать офсетная. Усл. печ. л. 46,44.

Тираж 2000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0307-5

© Шутенко Ю. Т., 2008
© Оформление, издательство "БХВ-Петербург", 2008

Оглавление

- Благодарности 11
- Введение 13
- Глава 1. HTML..... 17
 - Что представляет собой документ HTML? 17
 - Кто в DOM-ике живет? 19
 - О пользе первичного ключа 30
 - Работа с объектами. Сборка произвольного HTML-файла 31
 - Принципы создания элементов документа 32
 - Сказ про овечку Долли в доме HTML 36
 - Практическое использование 42
 - Отчеты в формате HTML/XHTML 42
 - Некоторые нюансы использования HTML DOM с VFP 44
- Глава 2. XML и другие X..... 47
 - Терминология..... 47
 - Well-formed против valid 48
 - Структура XML-документа 49
 - Пролог 50
 - Корневой элемент 52
 - Элементы 53
 - Типы структуры XML-документов (или смерть буриданова осла) 55
 - Экспорт из Visual FoxPro в XML 58
 - Общие правила создания объектов объектной модели XML-документа 58
 - Создание инструкции по обработке (processing instruction) 59
 - Особенности использования декларации *standalone* 61
 - XSLT-преобразование 64

Глава 3. JavaScript, AJAX, JSON	77
JavaScript — давайте познакомимся!	77
Элемент <i>script</i>	78
Где место JavaScript в документах?	80
Эти "переменчивые" переменные	83
Двуликий оператор <i>var</i>	85
Массивы	87
Функции	90
Объекты	94
JSON и AJAX	99
 Глава 4. ASP.NET + Visual FoxPro	 105
Терминология	105
Работа с базами данных Visual FoxPro	106
Web-сайты на ASP.NET	107
Создание простейшего сайта	108
Работа с локальными данными	122
Работа с данными, размещенными на публичном сервере	139
 Глава 5. Web-службы	 147
Использование Web-служб	149
Получение информации через сервисные приложения VFP	149
Получение информации с помощью XML DOM	160
Получение информации с помощью <i>WSDLReader</i>	165
Создание собственной Web-службы	169
 Глава 6. Введение в LINQ	 181
Первые шаги	182
Что требуется для работы с LINQ?	182
Первый проект с LINQ	183
Подготовка	183
Работа с ORD	187
Создание страницы с данными из таблиц	190
 Глава 7. Введение в SilverLight	 203
Что такое SilverLight?	203
Что требуется для введения SilverLight в приложения Visual FoxPro?	204
Первые шаги	204
Где разместить вызов метода создания плагина?	210
Где разместить вызов функции создания плагина?	211
Файл содержания SilverLight	212
Объект <i>Canvas</i>	212
Размещение объекта <i>Canvas</i>	213
Первые результаты	214

Глава 8. COM, DCOM и COM+	221
Краткий экскурс в историю	221
Что такое COM?	222
Что такое DCOM?	223
Что такое COM+?	224
"Состояние без состояния"	224
Потоки и апартаменты	225
COM+ и транзакции	226
Как выполнять транзакции под COM+ для данных VFP	228
Создание классов <i>CRMWorker</i> и <i>CRMCompensator</i>	229
Глава 9. <i>FileSystemObject</i>	237
Объект <i>FileSystemObject</i>	238
Свойство <i>Drives</i>	238
Методы <i>FileSystemObject</i>	239
Объект <i>File</i>	240
Свойства объекта <i>File</i>	240
Методы объекта <i>File</i>	242
Объект <i>Folder</i>	243
Свойства объекта <i>Folder</i>	243
Методы объекта <i>Folder</i>	247
Объект <i>Drive</i>	248
Свойства объекта <i>Drive</i>	248
Объект <i>TextStream</i>	248
Свойства объекта <i>TextStream</i>	249
Методы объекта <i>TextStream</i>	250
Глава 10. Объекты <i>WshShell</i> и <i>WshNetwork</i>	253
Что такое WSH?	253
Объект <i>WshShell</i>	255
Свойства	255
Методы	262
Объект <i>WshNetwork</i>	268
Свойства	268
Методы	268
Глава 11. Использование Windows Management Instrumentation	273
Структура WMI	275
Создание объектов	276
Подключение с помощью WMI-моникера	277
Извлечение информации из объектов WMI	280
Провайдеры	280
Работа с классами	284
Особенности работы с WMI на Windows Vista	297

Глава 12. Использование ActiveX	301
Использование WEB Browser Control	301
Планировщик маршрута на базе Microsoft Virtual Earth.....	302
Как создать объект <i>VEMap</i>	303
Реализация	305
Использование Windows Image Acquisition	314
Глава 13. Сервер MySQL и Visual FoxPro	329
Краткие характеристики версии 5.0	330
Установка и конфигурирование MySQL сервера.....	332
Работа с мастером конфигурации MySQL сервера	334
Инструменты для работы с базами данных MySQL.....	346
Создание и работа с базой данных MySQL.....	347
Разметка типов данных MySQL в типы данных Visual FoxPro	353
Глава 14. Advantage Database Server 9.0	357
Что пишут разработчики сервера о своем продукте?	357
Чем он может быть интересен для программистов Visual FoxPro?	359
Установка Advantage Database Server.....	361
Начало работы.....	366
Утилита конфигурации	366
Утилита Advantage Data Architect (ARC)	367
Конфигурация Advantage серверов и клиентов	370
"Кто есть кто" в Advantage?	370
Как сконфигурировать?	371
Конфигурация Advantage Local Server	371
Конфигурация Advantage Database Server	373
Конфигурация клиента	374
Проба пера.....	381
Работа со свободными таблицами Visual FoxPro	381
Импорт базы данных Northwind в словарь данных Advantage	384
Использование сквозных запросов	389
Полнотекстовый поиск (FTS).....	391
Использование утилиты DBCCConvert.prg	393
Заключение	397
Глава 15. Использование <i>CursorAdapter</i>	399
Класс <i>CursorAdapter</i>	400
Создание настраиваемого класса <i>CursorAdapter</i>	401
Настройка класса для работы с MySQL через ODBC	405
Как насчет модификации команды <i>SelectCmd</i> ?	414
Глава 16. <i>XMLAdapter</i>	423
Как извлечь данные из DataSet Web-службы?	426
Особенности работы с вложенными таблицами	433

Извлечение XML-данных из MS SQL Server.....	434
MS SQL Server 2000	434
Как извлечь данные?	435
MS SQL Server 2005	441
Особенности формирования XML MS SQL Server	449
Глава 17. Расширение Visual FoxPro с помощью Visual FoxPro	453
IntelliSense	456
Объекты-компаньоны	464
Глава 18. Инструментарий Visual FoxPro	469
Task List (Список задач)	469
Environment Manager (Диспетчер среды)	476
Data Explorer (Проводник к данным)	482
Работа с природными базами Visual FoxPro	482
Глава 19. Полезные решения	493
Предотвращение повторного запуска приложения	493
Поиск окна приложения	493
Использование таблицы атомов	505
Использование <i>Mutex</i>	511
ПРИЛОЖЕНИЯ	517
Приложение 1. <i>FileSystemObject</i> и WMI	519
Объект <i>Scripting.FileSystemObject</i>	519
Настройки безопасности WMI	542
Язык запросов WMI Query Language	546
Приложение 2. Свойства и методы объекта <i>WshShell</i>	549
Набор переменных среды — <i>PROCESS</i>	549
Методы объекта <i>WshShell</i>	550
Специальные каталоги	555
База данных foxdevcons (MySQL)	558
Приложение 3. Описание компакт-диска	561
Предметный указатель	565

***МОЕЙ ЛЮБИМОЙ ЖЕНЕ ИРОЧКЕ,
СТОЙКО ПЕРЕНОСЯЩЕЙ ВСЕ ТЯГОТЫ ЖИЗНИ С ПРОГРАММИСТОМ,
И ЛЮБИМОЙ ДОЧУРКЕ КСЮШЕ ПОСВЯЩАЮ ЭТУ КНИГУ***

Благодарности

Неоценимую помощь при создании этой книги оказали Александр Медведев (Серпухов) и Сергей Сизов (Москва), которые смогли найти время для вычитки и редактирования глав.

Хотел бы поблагодарить

- ❑ Вадима Пирожкова,
- ❑ Равиля Тухватулина,
- ❑ Вячеслава Клепинина,
- ❑ Татьяну Агафонову

за дельные советы и моральную поддержку.

Особую благодарность хотел бы выразить

- ❑ Алану Гриверу (Yair Alan Griver),
- ❑ Алексею Цингаузу,
- ❑ Крейгу Бернтсону (Craig Berntson)

за некоторые идеи, которые были использованы при написании книги.

Введение

Данные в современном мире представлены в виде различных форматов: текстовых файлов, баз данных различных серверов, реестров, HTML-документов, XML-документов, сообщений e-mail, в виде содержания сообщений протокола доступа к простым объектам (SOAP), файлов приложений Microsoft Office, документов Word, Excel, массивов, чертежей, графиков и ... список этот велик.

Каждый конкретный формат данных обслуживается программами, которые могут общаться с файлами этого конкретного формата, а именно: читать, изменять, удалять и т. д. Сегодня очень часто требуется, чтобы пользователь конкретного приложения мог обращаться к данным различных форматов, работа с которыми не свойственна программному языку, на котором написано это конкретное приложение. Можно с полной уверенностью сказать, что не существует программы или приложения, которые могли бы работать со всеми форматами сразу. Однако существуют технологии, которые позволяют различным приложениям работать с данными различных форматов.

Visual FoxPro представляет собой систему, удачно соединяющую в себе способность великолепной работы с базами данных, свободными таблицами и представлениями своего формата, а также возможность работать с данными различных форматов посредством OLE Automation (далее по тексту — Automation), который представляет собой формальный механизм межпроцессного общения, основанный на компонентной объектной модели — COM. С помощью Automation Visual FoxPro может обмениваться данными с различными приложениями, которые предназначены для работы с определенными для таких приложений форматами. Более того, приложение Visual FoxPro может само выступать в роли COM-сервера, благодаря чему другие приложения могут обращаться к данным, содержащимся в базах данных, свободных таблицах и представлениях Visual FoxPro.

Интернет уже давно играет важную роль в повседневной жизни. Ежедневно создаются и публикуются для свободного доступа десятки тысяч документов.

Документы в Интернете можно рассматривать двояко: как источник информации и как способ представления информации. В качестве основы этих документов используется язык разметки: для HTML-документов — язык разметки гипертекста; для XML-документов — расширяемый язык разметки. Знание этих весьма несложных формальных языков поможет разработчику по-новому взглянуть на представление данных. Совсем необязательно, чтобы такие документы были опубликованы в Интернете. Они с успехом могут использоваться в локальных сетях для более привлекательного представления данных, а также для обмена данными между разными приложениями. В этой книге будет показано, как получать данные из такого рода документов, а также как использовать данные для создания, например, очень сложных интерактивных отчетов, которые практически невозможно сделать с помощью стандартных средств Visual FoxPro.

В последнее время все большее распространение получает технология ASP.NET, в основе которой лежит "web application framework", созданный корпорацией Microsoft для того, чтобы программисты могли создавать динамические Web-сайты, Web-приложения и Web-сервисы. ASP.NET является дальнейшим развитием технологии ASP — активных серверных страниц. Знание этой технологии, особенно в части работы с данными, содержащимися в базах данных и таблицах, поможет разработчикам приложений Visual FoxPro выводить данные в Интернет и, кроме того, быть на переднем фронте инноваций. Технология ASP.NET позволяет использовать подготовленные ранее HTML-документы без какой-либо их коррекции или переделки. XML-документы являются частью этой технологии. Они используются как для целей конфигурации, так и в качестве источников данных.

Во всех, без исключения, приложениях очень важную роль играет производительность. В системах управления базами данных — это скорость выполнения запросов. В Интернете — это скорость загрузки ресурсов. Разумеется, чем меньше по объему ресурс, тем быстрее он будет загружен. А что делать в случае, когда пользователю требуется вывести огромный объем данных? Здесь на помощь придет технология AJAX, благодаря которой вы можете извлекать порции данных и пополнять их, при необходимости, по запросам пользователя. Это сродни настраиваемому извлечению данных в Visual FoxPro, хотя для этого потребуются некоторые знания языка JavaScript. Вы сможете получить их, прочитав соответствующую главу настоящей книги. Это позволит вам создавать скрипты, которые будут выполнять множество операций по обслуживанию данных, и это можно сделать с помощью Visual FoxPro, создав соответствующие классы.

Несмотря на то, что Visual FoxPro имеет огромное количество команд и функций, использование технологий Windows позволяет упростить общение с самой операционной системой, ее файловой системой и все это легко интегрируется в приложения, создаваемые на базе языка Visual FoxPro.

Размещение баз данных Visual FoxPro в Интернете осложняется тем, что большинство публичных серверов, на которых обычно размещаются сайты, в качестве операционной системы используют свободно распространяемые системы на основе спецификации UNIX, в то время как для размещения сайтов, данные для которых будут поставляться из баз данных Visual FoxPro, требуется серверная операционная система Windows. В связи с этим для разработчиков актуально использование в качестве back-end сервера такой популярной системы управления базами данных, как MySQL. Обмен данными между Visual FoxPro и MySQL возможен с помощью ODBC, однако необходимо учитывать разность форматов полей обеих систем управления базами данных и некоторые особенности языка SQL, используемого в MySQL.

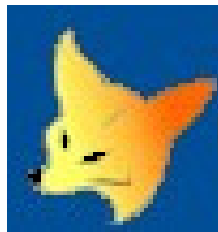
В конце 2007 года был анонсирован новый Advantage Database Server версии 9, выпускаемый Extended Systems Inc., дочерней компанией Sybase. Он позволяет использовать базы данных и таблицы Visual FoxPro без участия каких-либо посредников (OLEDB или ODBC) и может рассматриваться как одна из возможностей создания клиент-серверных приложений на основе баз данных Visual FoxPro. Финальный релиз этого продукта вышел в марте 2008 года.

Появившиеся в последних версиях Visual FoxPro классы `CursorAdapter` и `XMLAdapter` позволяют разработчикам выстроить мосты между различными форматами данных, серверами баз данных и XML-документами. Благодаря этим классам упрощается обмен данными между разного рода приложениями. Однако использование упомянутых классов вызывает множество вопросов у разработчиков, поэтому эти два класса будут рассмотрены более подробно.

Visual FoxPro уникальный язык программирования и его уникальность определяется не только и не столько огромным количеством функций и компонентов, но также возможностью расширения его функциональных возможностей за счет его самого. Ни один из языков программирования не имеет такой возможности. Богатый инструментарий Visual FoxPro, составляющий основу интерактивной среды разработки, дополнен множеством инструментов, размещенных в XSource, который Даг Хеннинг справедливо назвал "золотыми россыпями". И это действительно так. Используя исходный код, вы можете изменить эти инструменты, внося в них новые свойства и методы, настроить их "под себя" и облегчить тем самым свой труд программиста. Вы можете, используя в качестве шаблонов уже существующие "мастера", создавать свои уникальные микроприложения, передавая тем самым часть своей рутинной работы Visual FoxPro.

Разумеется, эта книга не сможет описать все возможности, предоставляемые разработчикам Visual FoxPro смежными технологиями, но я надеюсь, что она поможет вам начать их освоение.

ГЛАВА 1



HTML

Что представляет собой документ HTML?

Дословно HTML — HyperText Markup Language, переводится как "язык разметки гипертекста". Если взглянуть на исходный текст документа, то мы увидим, что HTML-документ представляет собой некий текст, размеченный тэгами и имеющий секции заголовка и тела (`head` и `body`). Если же взглянуть на этот же документ глазами программиста Visual FoxPro, то можно представить HTML-документ как некий контейнер, содержащий в себе другие контейнеры. В обычном HTML-документе имеется три типа контейнеров: контейнер самого документа, контейнер заголовка документа и контейнер тела документа.

Контейнер HTML-документа размещается между тэгами `<html>` и `</html>`. Первый из тэгов называется открывающим, второй — закрывающим. Этот контейнер не может содержать ничего, кроме двух других типов контейнеров — заголовка и тела документа. Открывающий тэг `<html>` может содержать атрибуты `lang` и `dir`, первый содержит информацию о первичном языке локализации документа, который по стандарту ISO идентифицируется двухсимвольным кодом языка, например, "ru", а второй атрибут указывает направление текста, идентифицируемое трехсимвольным обозначением "ltr" (слева направо) или "rtl" (справа налево). Оба атрибута могут быть опущены. По умолчанию используется значение "ltr".

Контейнер заголовка документа `head` обозначается тэгами `<head>` и `</head>`. Он содержит размеченную тэгами информацию о текущем документе, такую как титул документа, выводимый в заголовке окна обозревателя Интернета, ключевые слова, которые используются поисковыми системами, информацию об авторе, метаданные, документы и прочие элементы. В контейнере за-

головка документа обычно размещают каскадные таблицы стилей (CSS) или ссылки на внешние файлы, содержащие такие таблицы. Кроме CSS в заголовке документа обычно размещают тексты скриптов JavaScript или VBScript. Часть элементов мы рассмотрим далее, а полную информацию об элементах, которые может содержать заголовок, вы можете получить на сайте W3-консорциума (<http://www.w3.org>), занимающегося разработкой стандартов для Web. Отметим, что все, что размещено в контейнере заголовка документа является служебной информацией документа и скрыто от посетителя ресурса, когда HTML-документ выводится пользователю для просмотра с помощью какого-либо интернет-обозревателя: Internet Explorer, Opera, FireFox, Safari и пр.

Ну и, наконец, контейнер тела документа, обозначаемый тэгами `<body>` и `</body>`. В этом контейнере размещается основное содержание документа, которое будет отображено пользователю. Содержание документа может включать в себя текст, разделы, таблицы, картинки, ссылки на другие ресурсы, медиаэлементы и пр. Содержание документа размечено тэгами. Например, параграфы заключаются в тэги `<p>` и `</p>`, картинки — в тэги `` и ``, секции документа — в тэгах `<div>` и `</div>` и т. д.

Хотя стандарт HTML-документа определяет, что закрывающие тэги некоторых элементов можно опустить, но все-таки лучше их использовать попарно, если не предусмотрено иное. Например, тэг переноса строки, `
`, не имеет закрывающего тэга. Это же справедливо и для тэгов `<input>`, закрывающий тэг которых просто запрещен.

Тэги разметки гипертекста могут иметь атрибуты, и число атрибутов может быть достаточно большим. Часть атрибутов выделены и определены как глобальные. Глобальные атрибуты отличаются от прочих тем, что присущи всем элементам HTML-документа. Одним из важнейших атрибутов является атрибут `id`, который в HTML-документе имеет такое же значение, как и первичный ключ для таблицы в Visual FoxPro. К сожалению, многие Web-дизайнеры практически не используют этот атрибут с целью идентификации элементов HTML-документа, а предпочитают использовать с целью идентификации атрибут `name`. Однако я советую использовать этот атрибут повсеместно. В полезности этого правила вы убедитесь сами, при описании разбора HTML-документа, а также при описании оформления для более привлекательного отображения. Атрибуты тэгов разметки можно было бы сравнить со свойствами объектов Visual FoxPro, и такое сравнение было бы правильным, за исключением одного момента — в качестве атрибута тэга может выступать событие. Использованием объектных ссылок в качестве свойств объектов нас не удивит, но то, что в качестве свойства может выступать событие, это для вас, скорее всего, нечто новое.

В Visual FoxPro контейнеры могут содержать другие контейнеры. Это справедливо и для HTML-документа. Так, например, контейнер тела документа может содержать вложенный контейнер формы, контейнер формы может содержать вложенный контейнер таблицы, контейнер таблицы может содержать вложенные контейнеры заголовка и тела таблицы и т. д. Что еще могут содержать перечисленные контейнеры? Коллекции элементов. Каждая коллекция содержит однотипные элементы. Например, коллекция таблиц содержит все таблицы, которые были определены в документе. Тело таблицы содержит коллекцию строк таблицы и коллекцию ячеек таблицы. И так далее, и тому подобное. Кроме того, существует глобальная коллекция всех объектов HTML-документа.

Чем являются контейнеры в Visual FoxPro? Объектами. Что содержат контейнеры в Visual FoxPro? Объекты. Можем ли мы применить такую же схему к HTML-документу? Безусловно. В первом приближении HTML-документ можно сравнить с формой Visual FoxPro.

Итак, HTML-документ можно рассматривать как набор объектов. Возникает резонный вопрос — как добраться до этих объектов? Здесь нашим помощником будет объектная модель документа, в английском варианте — Document Object Model (DOM).

Кто в DOM-ике живет?

DOM представляет собой объектную модель, которая описывает связь элементов HTML-документа с высшим элементом структуры — объектом `document`. Зная структуру документа, мы можем обращаться к его элементам, используя их имена в объектной модели.

DOM разделена на три части — Core, XML и HTML и на три уровня — 1, 2 и 3. В данной главе мы будем рассматривать HTML DOM. DOM представляет HTML-документ в виде древовидной структуры, содержащей элементы, атрибуты и текст. Согласно DOM, все, что имеется в HTML-документе, представляет собой набор узлов. Узел представляет единичный элемент в дереве документа. Опишем правила DOM.

- ☐ Документ в целом представляет собой узел документа.
- ☐ Каждый тэг документа представляет собой узел элемента.
- ☐ Текст, содержащийся в HTML-элементах, представляет собой узел текста.
- ☐ Любой атрибут HTML-элемента представляет собой узел атрибута.
- ☐ Любой комментарий представляет собой узел комментария.

Попробуем перевести все это на язык Visual FoxPro.

- ❑ Документ в целом представляет собой объект документа.
- ❑ Каждый тэг документа представляет собой объект коллекции однотипных элементов.
- ❑ Текст, содержащийся в HTML-элементах, представляет собой значение свойства объекта элемента.
- ❑ Любой атрибут HTML-элемента представляет собой объект коллекции атрибутов.
- ❑ Любой комментарий представляет собой объект коллекции комментариев.

Нужно отметить, что говорящим по-русски программистам повезло со словом DOM. Прочтите это сокращение по-русски вслух. Что получилось? Дом! А кто в доме живет? Семья. А теперь представьте себе цепочку наследования:

Родитель → дети → внуки → правнуки → праправнуки и т. д.

А если добавить в эту семью королевской крови, можно говорить о королевской цепочке наследования:

Родитель → первый сын → первый внук → первый правнук → первый праправнук и т. д.

А если у первого, т. е. старшего, дитя есть братья и сестры? Так это уже родственники, по-английски — "siblings". Но так ли все просто с родством в HTML-документе?

Разберем **на основе родственных связей** простейший HTML-документ, исходный текст которого представлен в листинге 1.1.

Листинг 1.1. Текст шаблона HTML-документа

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>HTML DOM для FOXers</title>
<meta name="Author" content="John Doe">
</head>
<body id="example1">
<form id="form1" name="test">
<h1 id="header1">Это заголовок</h1>
<p id="p1">Это <b>параграф</b>.</p>
</form>
</body>
</html>
```

Для исследования этого текста нам потребуется объект HTML-документа. Создайте новый программный файл Visual FoxPro и напечатайте строки кода, приведенные в листинге 1.2, или откройте на редактирование файл `htmldom1.prg`, размещенный в каталоге `JSVFPBook\Examples\Glava1` прилагаемого к книге диска.

ЗАМЕЧАНИЕ

В объектной модели документа принято использовать так называемую *camel-нотацию*. Это позволяет при поиске информации избежать некоторой путаницы в различии одноименных объектов и свойств, используемых в различных языках. Использование *dot-нотации*, т. е. разделение цепи наследования с помощью точки, присуще всем объектным языкам.

Листинг 1.2. Программное создание шаблона и объекта шаблона HTML-документа

```
Local ;
    lcHtmlTemplate As String,;
    loHtmlDocument As MSHTML.DOMDocument,;
    loVisualRoot As MSHTML.HTMLDocument,;
    loNode as MSHTML.HTMLHtmlElement

lcHtmlTemplate=;
[<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">]+;
[<html>]+;
[<head>]+;
[<title>HTML DOM для FOXers</title>]+;
[<meta name="Author" content="John Doe">]+;
[</head>]+;
[<body id="example1">]+;
[<form id="form1" name="test">]+;
[<h1 id="header1">Это заголовок</h1>]+;
[<p id="p1">Это <b>параграф</b>.</p>]+;
[</form>]+;
[</body>]+;
[</html>]

loHtmlDocument=Createobject("htmlfile")
```

Полученный нами объект `loHtmlDocument` представляет собой пустой объект документа.

ПРЕДУПРЕЖДЕНИЕ

Попытка посмотреть содержимое полученного пустого объекта в отладчике Visual FoxPro неминуемо приведет к исключению C5.

Этот объект является корневым узлом для всех узлов документа, которые будут введены в него. Внесите в полученный объект HTML-документа шаблон, подготовленный в переменной `lcHtmlTemplate`, для чего дополните код редактируемой программы представленной далее строкой:

```
loHtmlDocument.Write(lcHtmlTemplate)
```

Теперь мы можем обратиться к объекту HTML-документа и исследовать его свойства. Дополните исходный текст программы строками программы, представленными далее:

```
? "Информация о корневом узле документа: "  
? "тип документа: "  
?? loHtmlDocument.doctype  
? "название узла документа: "  
?? loHtmlDocument.nodeName  
? "имеются ли у этого узла дочерние элементы? "  
?? loHtmlDocument.hasChildNodes  
? "Сколько дочерних элементов имеется у этого узла? "  
?? loHtmlDocument.childNodes.length
```

Свойство `doctype` возвратит `.Null.`, но здесь нет ошибки. Это верно для всех HTML-документов.

Хотя в первой строке шаблона HTML-документа использована декларация типа документа, она в HTML-документах носит характер комментария, в чем нетрудно убедиться. Так как декларация типа документа находится в первой строке разбираемого HTML-документа, то в силу этого она является первым дочерним элементом корневого узла и получить объектную ссылку на объект узла можно, используя цепочку наследования: *родитель* → *первый дочерний узел*.

Дополните текст программы приведенными далее строками:

```
loDoctype=loHtmlDocument.firstChild  
? "Информация о декларации типа документа: "  
? "название узла документа: "  
?? loDoctype.nodeName  
? "тип узла документа: "  
?? loDoctype.nodeType  
? "содержание узла документа: "  
?? loDoctype.text
```

В терминах DOM вторым дочерним узлом разбираемого HTML-документа является узел `html`. Этот узел, в свою очередь, является родительским узлом для всех остальных узлов HTML-документа. Чтобы добраться до него на основе родственных связей, нужно вспомнить, что он является вторым и одно-

временно последним дочерним элементом разбираемого HTML-документа, т. е. ближайшим родственником первого узла.

Цепочка наследования может в данном случае иметь два варианта:

```
loHTML=loHtmlDocument.firstChild.nextSibling  
loHTML=loHtmlDocument.lastChild
```

Теперь вернемся к исходному тексту примера. У элемента `html` есть два своих дочерних элемента: `head` и `body`. Думаю, что вам теперь не составит труда перебрать все способы получения объектной ссылки на эти элементы в разбираемом HTML-документе на основе родственных связей.

Варианты получения объектной ссылки на элемент `head` через цепочки наследования от корневого узла:

```
loHead=loHtmlDocument.firstChild.nextSibling.firstChild
```

или

```
loHead=loHtmlDocument.lastChild.firstChild
```

А теперь с помощью уже знакомых вам схем получим объект узла метаданных документа примера, который несет информацию об авторе документа. Элемент `meta` является вторым и одновременно последним дочерним узлом узла `head`, значит для получения объектной ссылки на элемент `meta` нужно просто удлинить цепочку наследования, полученную для родительского узла `head`:

```
loMeta=;  
loHtmlDocument.firstChild.nextSibling.firstChild.firstChild.NextSibling
```

или

```
loMeta=;  
loHtmlDocument.firstChild.nextSibling.firstChild.lastChild
```

В нашем простейшем примере имеется узел параграфа, в котором слово "параграф" выделено жирным с помощью тэгов ``. Поскольку HTML DOM указывает, что каждый тэг документа представляет собой узел документа, то следовательно, узел параграфа может иметь дочерний узел. Проверим. Если перебирать узлы, начиная от узла документа, то, при условии знания содержания документа, у нас выстроится длинная цепочка:

```
loParagraph=;  
loHtmlDocument.lastChild.lastChild.lastChild.lastChild
```

которая может быть еще длиннее, если использовать королевскую родственную нотацию. Убедимся, что получили объектную ссылку на объект параграфа HTML-документа:

```
? loParagraph.nodeName
```


Проверим, есть ли у параграфа дочерние узлы:

```
? loParagraph.hasChildNodes
```

Узлы у параграфа имеются, и мы можем узнать их количество:

```
? loParagraph.childNodes.length
```

Последняя команда выведет на экран может быть довольно неожиданный для вас результат: два! Если следовать второму правилу DOM, то одним из узлов должен бы быть узел тэга ``, но тогда возникает резонный вопрос: а кто второй? Это очень просто узнать:

```
? loParagraph.firstChild.nodeName
```

```
? loParagraph.lastChild.nodeName
```

Полученный результат, может быть, удивит вас еще больше, т. к. имена двух узлов совпадают и имеют значение `#text`. Если мы исполним приведенные далее четыре командные строки:

```
? loParagraph.firstChild.nodeType
```

```
? loParagraph.firstChild.nodevalue
```

```
? loParagraph.lastChild.nodeType
```

```
? loParagraph.lastChild.nodevalue
```

то увидим, что мы имеем дело с узлами типа "текст" и нам будет выведен разорванный на части текст, а именно слово "Это" и символ "точка". А куда пропало слово "параграф"? И что случилось с тэгом ``? Давайте разбираться с потомством нашего параграфа. Есть подозрения, что он многоженец!

1. Можем ли мы подсчитать число прямых потомков параграфа? Несомненно:

```
? loParagraph.childNodes.length
```

Ответ: два. С этими детьми разобрались. Это два текстовых узла.

2. Может быть первый дочерний узел имеет своих потомков?

```
? loParagraph.firstChild.hasChildNodes
```

Ответ: нет.

3. А что это за группа детишек `children` спряталась в свойствах? Сколько их?

```
? loParagraph.children.length
```

Ответ: один.

4. Как зовут дитя?

```
? loParagraph.children(0).nodeName
```

Ответ: B.

5. А как насчет его содержания?

```
? loParagraph.children(0).innerText
```

Ответ: параграф.

Что мы имеем в итоге? Разорванный и распределенный по элементам текст. Нужно ли такой разорванный текст как-то склеивать? К счастью нет, т. к. мы можем получить его в нормальном виде с помощью обращения к свойству `innerText` узла параграфа:

```
? loParagraph.innerText
```

Давайте окончательно разберемся с потомством нашего параграфа. В имплементации HTML DOM имеется две коллекции дочерних элементов:

☐ `childNodes`;

☐ `children`.

При обращении к первой из них извлекается коллекция HTML-элементов и объектов `TextNode`, которые являются прямыми наследниками указанного объекта, включая пустые текстовые узлы и комментарии.

ПРИМЕЧАНИЕ

Internet Explorer и iCab не подсчитывают пустые текстовые узлы. В принципе это правильное поведение.

При обращении ко второй извлекаются объекты DHTML, которые являются прямыми наследниками указанного объекта. Иначе говоря, вторая коллекция возвращает только те узлы, которые являются узлами элемента.

То есть в рассмотренном случае тэг `` является внутренним узлом параграфа, но не является дочерним узлом. Будьте внимательны, при работе с этими коллекциями, т. к. разница между ними несколько размыта по определению! Например, полученный в программе объект документа `loHtmlDocument` не имеет среди свойств этой коллекции.

ПРИМЕЧАНИЕ

Если вы будете проверять коллекцию `childNodes` элемента, созданного через стандартный HTML-синтаксис, то вы найдете объекты `TextNode` в совершенно неожиданных и неподобающих местах, например, вместо разделителей строк. Однако если вы создаете элемент через Document Object Model (DOM), то, например, Windows Internet Explorer не будет создавать лишних объектов `TextNode`.

В приведенном ранее примере программы использовалось свойство `nodeType`, которое определяет тип узла HTML-документа и может возвращать значение от 1 до 12:

- ❑ `ELEMENT_NODE (1);`
- ❑ `ATTRIBUTE_NODE (2);`
- ❑ `TEXT_NODE (3);`
- ❑ `ENTITY_REFERENCE_NODE (5);`
- ❑ `ENTITY_NODE (6);`
- ❑ `COMMENT_NODE (8);`
- ❑ `DOCUMENT_NODE (9);`
- ❑ `DOCUMENT_FRAGMENT_NODE (11);`
- ❑ `NOTATION_NODE (12).`

Знание значений этого свойства оказывает большую помощь при программном лексическом разборе или обработке HTML-документа, когда требуется выбрать или удалить из документа только определенные элементы, например, выбрать только текстовые узлы или удалить закомментированные элементы.

Итак, с родственными связями, я думаю, вы разобрались, их использование оправдано в некоторых ситуациях. Например, в своем приложении WS2006 я использую цепочки наследования в редакторе страниц для вывода структуры HTML-документа в дереве документа, что делает процесс редактирования более удобным, т. к. я могу скрыть или, наоборот, развернуть секции документа или его отдельные узлы.

Вернемся к нашему примеру и рассмотрим его разбор с использованием коллекций.

ЗАМЕЧАНИЕ

Индексирование элементов коллекций в объектной модели документов HTML основано на значении 0, а не 1, как принято в Visual FoxPro.

Для того чтобы получить тот же самый элемент `<!DOCTYPE`, мы можем обратиться к коллекции `childNodes`, которая для объекта HTML-документа имеет только одно свойство `length` и только один метод `item()`.

Дополните текст программы приведенными далее строками:

```
loDoctype=loHtmlDocument.childNodes.item(0)
? "Информация о декларации типа документа: "
? "название узла документа: "
?? loDoctype.nodeName
? "тип узла документа: "
?? loDoctype.nodeType
? "содержание узла документа: "
?? loDoctype.text
```

Что касается узла `html`, то объектную ссылку на него можно получить, используя элемент с индексом равным единице в коллекции `childNodes`, т. к. мы знаем это из текста примера. Но это не единственный способ добраться до данного узла. Во-первых, мы можем использовать глобальную коллекцию всех элементов HTML-документа `all`, во-вторых, использовать коллекцию, которую можно получить с помощью метода `getElementsByTagName()` HTML-документа, а в-третьих, мы можем использовать специальное свойство объекта HTML-документа `documentElement`. Последний способ самый короткий, но он применим только к этому узлу.

Перебор членов коллекции `childNodes` может основываться либо на типе узла, либо на его имени. Дополните текст программы, приведенной в листинге 1.2, следующими строками:

```
? "Получение объектной ссылки на узел HTML:"
? "через члена коллекции дочерних узлов childNodes:"
loHTML=loHtmlDocument.childNodes.Item(1)
? "Название узла документа: "
?? loHTML.nodeName
? "через перебор членов коллекции и определение типа узла:"
For Each loNode In loHtmlDocument.childNodes
    If loNode.nodeType=1
        loHTML=loNode
        ? "название узла документа: "
        ?? loHTML.nodeName
        Exit
    Endif
Endfor
For Each loNode In loHtmlDocument.childNodes
    If loNode.nodeName="HTML"
        loHTML=loNode
        ? "тип узла документа: "
        ?? loHTML.nodeType
        Exit
    Endif
Endfor
```

При использовании метода `getElementsByTagName()` мы получаем объект коллекции одноименных тэгов. Хотя в HTML-документе тэг `<html>` может быть только один, нам все равно придется извлекать этот элемент из коллекции элементов через метод `item()`:

```
loHTML=loHtmlDocument.getElementsByTagName("html").item(0)
? loHTML.nodeName
```

При использовании свойства *documentElement* HTML-документа вы получите объектную ссылку на требуемый элемент *html*:

```
loHtml= loHtmlDocument.documentElement  
? loHTML.nodeName
```

В результате выполнения всех перечисленных команд вы получаете объектную ссылку на объект узла *html*. Обратите внимание на то, что имя узла выведено в верхнем регистре.

Теперь вернемся к исходному тексту примера. У элемента *html* есть два своих дочерних элемента: *head* и *body*. Варианты получения объектной ссылки на элемент *head* приведены далее:

```
loHead= loHtmlDocument.lastChild.childNodes.item(0)  
loHead=loHTML.childNodes.item(0)
```

ЗАМЕЧАНИЕ

Допускается использование в одном и том же документе нескольких элементов *head*. Подробнее об этом будет рассказано в главе, посвященной JavaScript.

ЗАМЕЧАНИЕ

Чтобы создать более безопасный код, используйте предварительную проверку на наличие дочерних узлов с помощью свойства *hasChildNodes*, которое имеется у каждого узла дерева HTML-документа.

А теперь с помощью уже знакомых вам схем получим объект узла метаданных документа примера, который несет информацию об авторе документа. Так как мы знаем из текста примера, что тэг *<meta>* является вторым тэгом в секции заголовка, то мы можем извлечь его из коллекции дочерних узлов с помощью индекса, равного единице:

```
loMeta=loHead.childNodes.item(1)
```

или из коллекции одноименных тэгов:

```
loMeta=loHtmlDocument.getElementsByTagName("meta").item(0)
```

Секция заголовка HTML-документа может содержать множество тэгов с именем *meta*, т. к. эти элементы определяют множество метаданных. Как получить объектную ссылку на желаемый объект коллекции одноименных тэгов? Достаточно легко! Нужно просто воспользоваться методом коллекции *namedItem()*. В нашем примере тэг *<meta>* несет информацию об авторе примера. Среди атрибутов этого тэга имеется атрибут с именем *name*, значение которого равно *Author*. Воспользуемся указанным методом:

```
loMetaCollection=loHtmlDocument.getElementsByTagName("meta")  
loMeta= loMetaCollection.namedItem("Author")  
? loMeta.nodeName
```

Этот узел имеет два атрибута: *name* и *content*. Значением первого мы воспользовались при получении объектной ссылки. Как получить значение второго атрибута? Рассмотрим работу с коллекциями атрибутов.

В случае наличия у узла атрибутов, у нас есть два варианта получения их значений: используя коллекцию узла `attributes` конкретного узла HTML-документа или непосредственно используя имена атрибутов.

Рассмотрим оба варианта получения значений атрибутов. Для получения объекта коллекции атрибутов воспользуемся приведенной далее строкой синтаксиса:

```
loMetaAttributes=loMeta.attributes
```

Далее HTML DOM начинает соседствовать (дома все-таки) с Core DOM. Core DOM имеет пять членов:

1. `getNamedItem()` (метод).
2. `item()` (метод).
3. `length` (свойство).
4. `removeNamedItem()` (метод).
5. `setNamedItem()` (метод).

Для получения значения конкретного атрибута мы можем воспользоваться первыми двумя членами модели, передав имя атрибута в качестве параметра. В случае использования первого члена модели синтаксис выглядит так:

```
? loMetaAttributes.getNamedItem("name").nodeValue  
? loMetaAttributes.getNamedItem("content").nodeValue
```

В случае использования второго члена модели синтаксис выглядит так:

```
? loMetaAttributes.item("name").value  
? loMetaAttributes.item("content").value
```

Однако гораздо проще использовать другой вариант получения значений атрибутов узла — с помощью прямого обращения к имени атрибута объекта:

```
? loMeta.name  
? loMeta.content
```

Теперь, когда у нас есть объектная ссылка на элемент `meta` и атрибут `content`, мы можем изменить значение атрибута с помощью простейшего синтаксиса:

```
loMeta.content="Juri Shutenko"  
? loMetaAttributes.item("content").value
```

Метод `setNamedItem` мы будем использовать в дальнейшем.