ЮРИЙ ШУТЕНКО

bhv®

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

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

Оглавление

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.

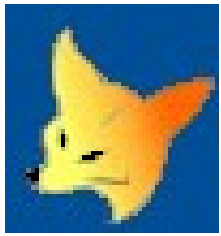
Размещение баз данных Visual FoxPro в Интернете осложняется тем, что большинство публичных серверов, на которых обычно размещаются сайты, в качестве операционной системы используют свободно распространяемые системы на основе спецификации UNIX, в то время как для размещения сайтов, данные для которых 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 Хенниг справедливо назвал "золотыми россыпями". И это действительно так. Используя исходный код, вы можете изменить эти инструменты, внеся 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, передав 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 `<img>` и `</img>`, 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, `<br>`, 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 удивишь, 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 `<b></b>`. 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 <b>, 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 <b>? 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 `<b>` 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.

Более подробно со свойствами и методами объектов HTML-документов вы можете ознакомиться в *приложениях 1* и *2*.

О пользе первичного ключа

Программистам Visual FoxPro ценность первичного ключа не нужно объяснять. С его помощью вы всегда однозначно идентифицируете нужную запись в таблице базы данных. В самом начале этой главы я писал, что большинство Web-дизайнеров игнорируют очень важный атрибут `id`, который можно установить практически для всех тэгов HTML-документа.

Предположим, что вам нужно разобрать какой-то HTML-документ с целью извлечения полезной информации, получить данные из какой-то таблицы, содержащейся в этом документе. Если таблица одна, то добраться до нее достаточно просто, используя метод объекта `getElementsByTagName()`. А если разбираемый файл содержит несколько таблиц, то вам придется сканировать коллекцию таблиц, чтобы найти нужную. При этом придется еще как-то анализировать полученный результат с целью узнать, получили ли мы желаемое.

Если вы установили для тэга идентифицирующий его атрибут `id`, то с помощью метода с именем `getElementById()` вы можете гораздо проще получить ссылку на требуемый вам объект. Этот метод принадлежит объекту `document` и в качестве параметра принимает значение атрибута `id` требуемого узла. То есть для того, чтобы получить объектную ссылку на нужный нам узел параграфа в разбираемом примере этой главы, нам нужно выполнить приведенную далее строку кода и убедиться, что мы имеем дело именно с узлом параграфа.

```
oParagraph=loHtmlDocument.getElementById("p1")
? oParagraph.nodeName
? oParagraph.innerText
```

> ***Примечание***
>
> Если вы случайно установили для некоторых элементов HTML-документа одинаковый ID, то указанный метод возвратит первый встретившийся элемент.

Вместо атрибута `id` в вызове этого метода может быть использован атрибут `name`, однако использование атрибута `id` предпочтительнее. Надо отметить, что современные средства редактирования HTML-документов производят автоматическую индексацию одноименных тэгов, если вы хотя бы однажды присвоили тэгу этот атрибут. Всегда используйте атрибут `id` — он позволит вам сэкономить массу времени. Если вернуться к разбираемому простейшему примеру, то вы, вероятно, заметили, что ни для раздела, ни для параграфа не определены никакие другие атрибуты, кроме идентификаторов.

Работа с объектами. Сборка произвольного HTML-файла

Для создания HTML-файла необходим объект HTML-документа. Его можно получить, используя COM-класс `htmlfile` с последующей загрузкой шаблона пустого документа, как это было сделано в листинге 1.2, или используя объект `InternetExplorer.Application`.

```
loHtmlDocument=CreateObject("htmlfile")
loHtmlDocument.write("имя_подготовленого_шаблона_файла")
```

или

```
loIEApplication= CreateObject("InternetExplorer.Application")
```

> ***Примечание***
>
> Полученный с помощью COM-класса `htmlfile` объект не имеет метода `Navigate()`, а вызов предлагаемого со стороны `IntelliSense` метода `createDocumentFromUrl()` в Visual FoxPro вызовет ошибку 1429. Поэтому используйте метод `write()` для записи начального содержания документа. В этом случае можно применять простейший шаблон, состоящий из трех обязательных элементов — `html`, `head` и `body`, или использовать заранее подготовленный шаблон с минимальным набором элементов и прописью некоторых метаданных в заголовке документа.

Листинг 1.3. Текст минимального шаблона для создания HTML-документов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>HTML Document that made by Visual FoxPro</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; windows-1251">
<META NAME="Author" CONTENT="Visual FoxPro Programmer">
<META NAME="Description" CONTENT="HTML Report from VFP">
</HEAD>
<BODY>
</BODY>
</HTML>
```

Предположим, что файл с таким содержанием имеет название htmlcommontemplate. Расширение имени файла шаблона может быть любым. Мы можем занести содержание этого файла в созданный объект с помощью двух строк кода:

```
lcInitialContent=FileToStr("путь\htmlcommontemplate.txt")
loHtmlDocument.write(lcInitialContent)
```

или в случае использования объекта `InternetExplorer.Application`:

```
loIEApplication/.Navigate("протокол://путь\htmlcommontemplate.txt")
loHtmlDocument=loIEApplication.document
```

Теперь объект имеет корневой узел с именем `#document`, свойства которого установлены в значения по умолчанию. Кроме того, корневой узел имеет два дочерних узла, первый из них содержит декларацию типа документа, а второй узел представляет собой элемент `html`. С помощью этого узла мы можем просматривать содержимое документа, обратившись к его свойству *`outerHtml`*.

```
lcHtmlContent= loHtmlDocument.childNodes.item(1).outerHtml
```

Используя это свойство, можно сохранять результат работы с объектом `htmlfile` во внешний файл с целью дальнейшего вывода полученного результата на просмотр посредством обозревателя Интернета.

```
=StrToFile(lcHtmlContent, " путь\имя_сохраняемого_файла.htm")
```

Принципы создания элементов документа

Новые объекты создаются двумя способами:

- через создание нового элемента в документе с последующим вызовом метода `appendChild()` или `InsertBefore()` целевого элемента с передачей ему объектной ссылки на созданный элемент;
- через создание дочерних элементов с помощью специальных методов целевых объектов.

> ***Примечание***
>
> Не все объекты HTML DOM имеют специальные методы создания дочерних элементов, поэтому преимущественным методом является использование первого способа. Однако если целевой элемент имеет специальный метод создания дочерних элементов, то использование первого варианта создания нового элемента может привести к нежелательным результатам, в частности к перекрытию (overlaping) элементов.

Поскольку нашей целью является наполнение документа желаемым содержанием, то нас больше всего интересует объект тела документа — `body`, т. к. внутри именно этого элемента размещается основное содержание документа. Чтобы получить объектную ссылку на элемент тела, нужно обратиться к одноименному свойству документа:

```
loBody=loHtmlDocument.body
```

Этот объект будет использоваться в качестве целевого как для добавления одиночных блочных элементов типа параграфа, так и элементов типа форм, секций и таблиц и им подобных.

После того как будет получена объектная ссылка на элемент `body`, мы можем сразу присвоить значения его атрибутам, которые являются частью глобальных атрибутов объектной модели документов. Имеет смысл присвоить значения только двум атрибутам — идентификатору элемента *`id`* и имени класса стилевой таблицы *`class`*. Если для атрибута *`id`* имеется одноименное свойство объекта, то значение атрибута *`class`* присваивается через свойство *`className`*. Для указания значений этих атрибутов используется dot-нотация: имя объекта и имя его атрибута разделяются точкой. То есть точно так же, как мы присваиваем значения свойствам объектов Visual FoxPro.

```
loBody.id="holder1"
loBody.className="forbrowser"
```

> ***Примечание***
>
> При присваивании значений атрибутам в программном коде Visual FoxPro имеется проблема с "правильностью" HTML-синтаксиса. Все значения атрибутов, представляющие собой литералы, по спецификации W3-консорциума должны быть заключены в двойные кавычки. Однако при работе с объектами в Visual FoxPro этого не происходит и при проверке созданного HTML-файла на странице проверки состоятельности документа **http://validator.w3.org** будут указаны ошибки в синтаксисе атрибутов. Имейте это в виду.

Начнем наполнять HTML-документ содержанием. Наиболее часто используемым элементом в HTML-документах является таблица. Объект `htmlfile` не имеет специального метода для создания таблиц, поэтому будем использовать первый вариант создания элементов: создадим объект таблицы в объекте `htmlfile` с помощью метода `createElement()`, а затем передадим его объекту `body`. Метод `createElement()` принимает единственный параметр, в качестве которого используется имя тэга создаваемого элемента. При вызове метода будут созданы оба тэга элемента — открывающий и закрывающий.

```
loNewTable=loHtmlDocument.createElement("table")
loTable= loBody.appendChild(oNewTable)
```

Так же, как и в случае создания объекта тела документа, атрибутам объекта таблицы, которые входят в состав глобальных атрибутов объектной модели документа, можно сразу присвоить значения:

```
loTable.id="layouttable"
loTable.className="layout"
```

Элемент таблицы в объектной модели документов может иметь четыре типа дочерних элементов — `caption`, `thead`, `tfooter` и `tbody`. В таблице разрешено иметь только по одному элементу первых трех типов. Что касается элемента `tbody`, то в таблице их может быть несколько и при добавлении этого элемента в таблицу он автоматически добавляется в коллекцию `tbodies`, которая

в объектной модели документов является членом объекта `table`. Элементы `thead`, `tfooter` и `tbody` представляют собой коллекции строк.

> ***Примечание***
>
> В спецификации W3-консорциума указывается, что элементы `thead`, `tfooter` и `tbody` должны иметь равное число колонок. Несоблюдение этого требования приведет к искаженному представлению таблицы в обозревателе Интернета.

Объект HTML-таблицы среди множества методов имеет три специальных метода для создания дочерних элементов:

- `createCaption()` — для создания общего заголовка таблицы;
- `createTFoot()` — для создания секции подножия таблицы;
- `createTHead()` — для создания головной секции, которую вы можете использовать для описания колонок таблицы;

но не имеет метода создания объекта тела таблицы `tbody`. Невольно возникает вопрос, как создать объект тела таблицы? Здесь имеется два варианта.

1. **Вы не планируете создавать секции заголовка и подножия таблицы.**

   В таком случае вы можете вообще не создавать объект `tbody`. Однако наличие коллекции подразумевает, что HTML-таблица может иметь несколько тел. Это действительно так и эту особенность HTML-таблиц можно очень широко использовать. Поэтому, если в таблице планируется использовать несколько объектов `tbody`, вы должны использовать первый вариант создания элементов с последующим добавлением их в качестве дочерних узлов объекта `loTable`. Разумеется, в этом случае наличие атрибута *`id`* становится просто необходимым, т. к. только с его помощью вы сможете получить объектную ссылку на требуемый для работы элемент тела таблицы.

2. **Вы планируете создать секции заголовка и подножия таблицы.**

   Если вы воспользуетесь методом `createTHead()` объекта `loTable`, то первый элемент `tbody` будет создан автоматически. Однако при использовании этого метода вы не получите объектной ссылки на объект тела таблицы. Ее придется создать вручную, выбрав первый элемент из коллекции `tBodies`. Если же вы планируете использовать в таблице несколько элементов `tbody`, то для удобства вы можете поступить следующим образом. Сначала вы получаете объектную ссылку на созданный методом `createTHead()` элемент `tbody` с помощью члена коллекции `tbodies` с индексом 0 и присваиваете атрибуту *`id`* элемента значимое строчное значение. Создание остальных элементов `tbody` проводится по первому варианту, а т. к. при добавлении нового элемента в таблицу вы автоматически получаете на него объектную ссылку, то сразу после добавления элемента в

таблицу присваиваете атрибуту *id* нужное значение. Тем самым вы будете уверены, что работаете с желаемым объектом.

Однако есть и другой вариант, который мы рассмотрим чуть позже, когда коснемся одной актуальной темы из биологии.

Итак, с учетом всего сказанного наполним содержанием таблицу `layouttable`. Эта таблица будет содержать все элементы, описанные в спецификации W3-консорциума.

Сначала создадим объект элемента `tcaption`. Все, что находится между открывающим и закрывающим тэгами этого элемента, будет отображено в HTML-странице над таблицей и отцентровано относительно таблицы.

```
loTCaption=loTable.createCaption()
loTCaption.innerText="This is a layout table!"
```

Далее создадим заголовочную секцию таблицы:

```
loTHead=loTable.createTHead()
```

Полученный объект имеет специальный метод `insertRow()`. Метод может принять один параметр — индекс, который определяет положение строки в коллекции `rows`. По умолчанию значение этого индекса равно –1, это означает, что новая строка будет добавлена в конец коллекции.

Вызов данного метода возвращает объект элемента строки таблицы, которые в свою очередь имеет специальный метод `insertCell()`, с помощью которого в элемент секции таблицы вставляется ячейка строки, являющаяся частью колонки таблицы.

Для ввода в секции таблицы `thead` и `tfooter` новых строк и ячеек вы должны вызвать оба метода столько раз, сколько строк и ячеек вы планируете иметь в таблице. Не забывайте сразу присваивать новым элементам атрибут *id*.

> ***Примечание***
>
> При использовании метода `insertRow()` в заголовочную секцию таблицы вместо элементов `th` будут вставлены элементы `td`. Различие между этими элементами заключается в использовании по умолчанию разных стилей. Текст в элементах `th` по умолчанию выделяется полужирным стилем, тогда как текст в ячейках `td` наследует стиль, определенный для документа в целом.

Немного иначе обстоит дело с элементом `tbody`. Этот элемент не имеет специального метода для вставки строк. Но таким методом обладает сама таблица. Вызов этого метода будет вставлять строки в элемент `tbody`, созданный при вызове метода `createThead()`. А если мы планируем иметь в таблице несколько элементов `tbody`? Но еще никто не отменял первого варианта создания элементов в DOM с помощью метода `createElement()` HTML-документа.

В таком случае в объекте документа сначала создается сам элемент `tbody` и размещается в таблице с помощью метода `appendChild()` объекта таблицы, в результате чего мы получаем его объектную ссылку. Теперь он может быть приемным отцом своих дочерних элементов — строк, которые, как и он сам, будут создаваться по первому варианту. Родители, детки... Определенно пора переходить к биологическим темам.

Сказ про овечку Долли в доме HTML

Предположим, что мы создали секцию таблицы `tbody` и внесли в нее 20 строк с пятью ячейками в каждой. Предположим также, что нам хотелось бы иметь еще одну такую же секцию, а может быть, и не одну. Но создание дополнительной секции будет проходить с участием объекта документа с последующим усыновлением, удочерением вложенных элементов. Как насчет клонирования? Весьма положительно, все-таки это не материальная субстанция!

Практически каждый элемент в HTML DOM имеет этот очень полезный метод, который может широко использоваться в нашей практике. Его имя `cloneNode()`. Этот метод способен значительно ускорить процесс создания HTML-документа.

Метод вызывается на клонируемом объекте и возвращает объектную ссылку на объект-клон, а пристроить ребенка к любому элементу-родителю мы уже умеем.

Всего делов-то:

```
loNewTBody=loTBody.cloneNode(.T.)
loTBody2=loTable.appendChild(loNewTBody)
loTBody2.Id="second_data_group"
```

А идентифицировать вложенные в клон элементы? Право, сущие пустяки! Кроме того, изменим цвет текста и фона ячеек, чтобы было легче различать две секции таблицы:

```
lni=1
For Each loTRow In loTBody2.rows
   lnj=1
   For Each loTCell  In loTRow.cells
      loTCell .className="blueonyellow"
      loTCell .Id="tb2r"+Alltrim(Str(lni))+"c"+Alltrim(Str(lnj))
      lnj=lnj+1
   Next
   lni=lni+1
Next
```

Что имеем в итоге? Две секции, которые различаются не только по атрибуту *id*, но имеют уникально идентифицированные вложенные элементы. Как насчет управления такой таблицей? Легко, т. к. объект HTML-документа имеет неоднократно упоминавшийся ранее `getElementById()`, которому в качестве параметра передается значение атрибута *id* любого элемента. Например, мы можем управлять отображением каждого элемента `tbody` таблицы. Для этой цели нам понадобится простенькая функция JavaScript. С данным языком вы познакомитесь поближе в *главе 3*.

В листинге 1.4 приведен полный код примера создания таблицы, кнопок управления и элемента `script`, который позволит управлять представлением таблицы. Чтобы вам было проще уяснить код примера, создание HTML-элементов и текст скрипта JavaScript прописаны явно. Я думаю, что вам не составит труда перевести некоторые действия по созданию элементов в пользовательские функции и процедуры Visual FoxPro.

Листинг 1.4. Пример создания HTML-файла с управляемой таблицей раскладки

```
Local loHtmlDocument As MSHTML.HTMLDocument, ;
   loContent As MSHTML.HTMLBaseElement, ;
   loBody As MSHTML.HTMLBody,;
   loTable As MSHTML.HTMLTable,;
   loTCaption As MSHTML.HTMLTableCaption,;
   loTHead As MSHTML.HTMLTableSection,;
   loTFoot As MSHTML.HTMLTableSection,;
   loTBody As MSHTML.HTMLTableSection,;
   loTBody2 As MSHTML.HTMLTableSection,;
   loTRow As MSHTML.HTMLTableRow, ;
   loTCol As MSHTML.HTMLTableCol, ;
   loTCell As MSHTML.HTMLTableCell,;
   loScript As MSHTML.HTMLScriptElement,;
   loComment As MSHTML.HTMLCommentElement

Local lcTemplate, lnExportResult, lcOutputFile
lcTemplate=;
   "C:\JSVFPBook\Examples\Glava1\commonhtmltemplate.htm"

loHtmlDocument=Createobject("htmlfile")
lcInitialContent=Filetostr(lcTemplate)
loHtmlDocument.Write(lcInitialContent)
loContent= loHtmlDocument.childNodes.Item(1)
? loContent.outerHtml
```

```
loBody=loHtmlDocument.body
loBody.Id="holder1"
loBody.className="forbrowser"
loNewTable=loHtmlDocument.createElement("table")
loTable= loBody.appendChild(loNewTable)
loTable.Id="layouttable"
loTable.Border="1"
loTable.className="layout"
loTCaption=loTable.createCaption()
loTCaption.innerText="This is a layout table!"

loTHead=loTable.createTHead()
loTRow=loTHead.insertRow()
For lni=1 To 4
   loTCol=loTRow.insertCell()
   loTCol.innerText=;
   "Колонка "+Alltrim(Str(lni)) + " строки заголовка"
Next

loTBody=loTable.tBodies.Item(0)
loTBody.Id="first_data_group"
For lni=1 To 5
   loTRow=loTable.insertRow()
   For lnj=1 To 4
      loTCol=loTRow.insertCell()
      loTCol.Id="tb1r"+Alltrim(Str(lni))+"c"+Alltrim(Str(lnj))
      loTCol.innerText=;
      "Колонка "+Alltrim(Str(lnj)) + " строки "+  Alltrim(Str(lni))
   Next
NEXT

loNewTBody=loTBody.cloneNode(.T.)
loTBody2=loTable.appendChild(loNewTBody)
loTBody2.Id="second_data_group"
lni=1
For Each loTRow In loTBody2.Rows
   lnj=1
   For Each loTCell  In loTRow.cells
      loTCell .className="blueonyellow"
      loTCell .Id="tb2r"+Alltrim(Str(lni))+"c"+Alltrim(Str(lnj))
      lnj=lnj+1
   Next
```

```
   lni=lni+1
Next

loNewButton=loHtmlDocument.createElement("button")
loButton1=loBody.appendChild(loNewButton)
loButton1.innerText="Спрятать 1-ю секцию"
loButton1.onClick="udf_HideTbody1()"
loNewButton=loHtmlDocument.createElement("button")
loButton2=loBody.appendChild(loNewButton)
loButton2.innerText="Показать 1-ю секцию"
loButton2.onClick="udf_ShowTbody1()"
loNewButton=loHtmlDocument.createElement("button")
loButton1=loBody.appendChild(loNewButton)
loButton1.innerText="Спрятать 2-ю секцию"
loButton1.onClick="udf_HideTbody2()"
loNewButton=loHtmlDocument.createElement("button")
loButton2=loBody.appendChild(loNewButton)
loButton2.innerText="Показать 2-ю секцию"
loButton2.onClick="udf_ShowTbody2()"
loNewScript=loHtmlDocument.createElement("script")
loScript=loBody.appendChild(loNewScript)
loScript.Type="text/javascript"

* в последующих девяти строках разбивка строки на части сделана
* исключительно с целью размещения на странице книги

loScript.Text=;
   'function udf_HideTbody1(){
document.getElementById("first_data_group").style.display="none"};'+;
   'function udf_HideTbody2(){
document.getElementById("second_data_group").style.display="none"};'+;
   'function udf_ShowTbody1(){
document.getElementById("first_data_group").style.display="block"};'+;
   'function udf_ShowTbody2(){
document.getElementById("second_data_group").style.display="block"}'

lnExportResult=;
   Strtofile(;
   loContent.outerHtml,;
   "C:\JSVFPBook\Examples\Glava1\example1.html";
   )
If lnExportResult>0
   Local loIEA As InternerExplorer.Application
```

```
    loIEA=Createobject("InternetExplorer.Application")
    loIEA.Navigate("C:\JSVFPBook\Examples\Glava1\example1.html")
    Do While loIEA.readyState !=4
       DoEvents
    Enddo
    loIEA.Visible=.T.
Endif
```

В результате выполнения этой программы вы получите управляемую таблицу, разные варианты отображения которой приведены на рис. 1.1—1.4.

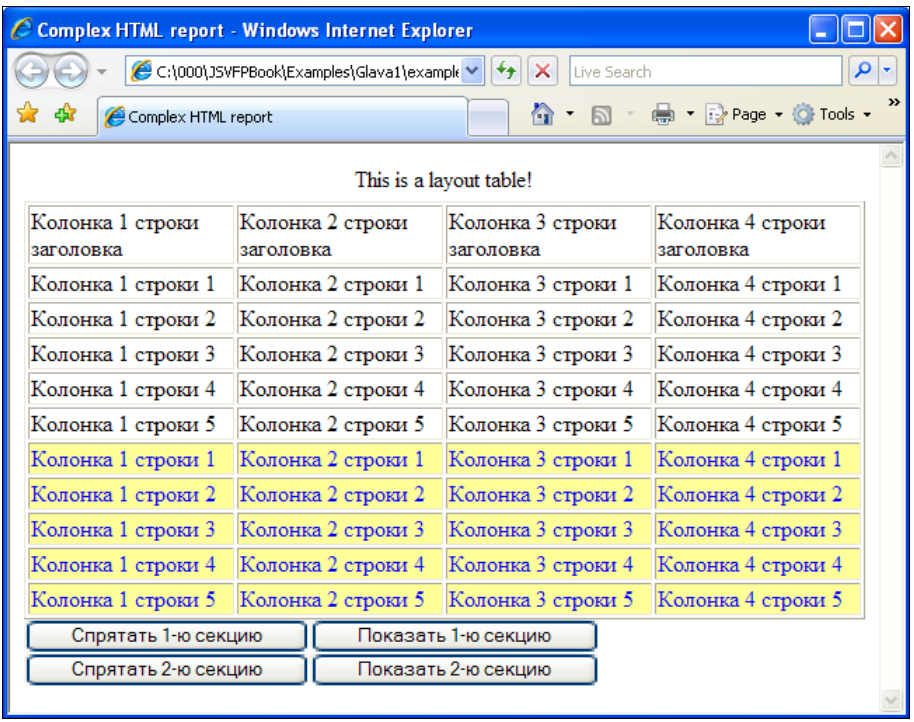


Рис. 1.1. Результат выполнения программы из листинга 1.4 (обе секции таблицы отображены)

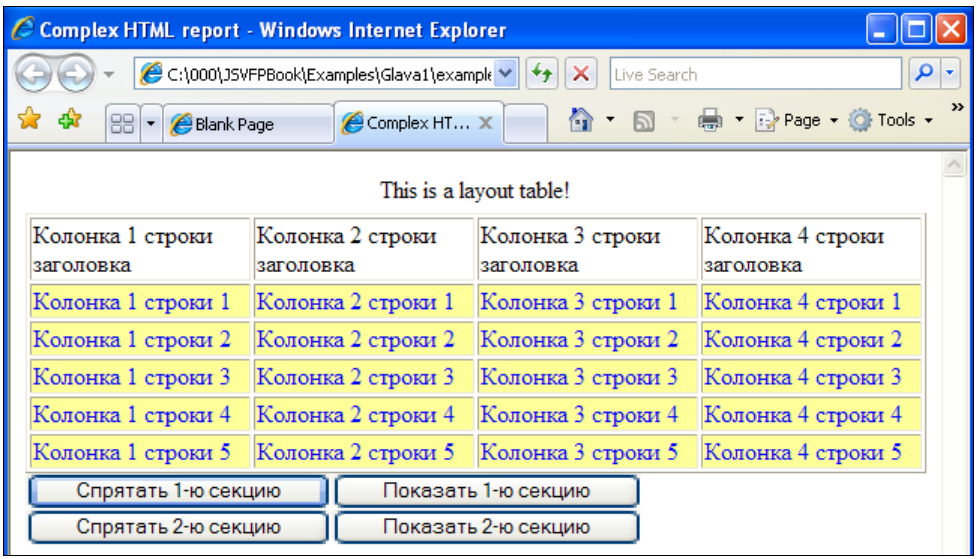


Рис. 1.2. Результат выполнения программы из листинга 1.4 (скрыта верхняя секция таблицы)

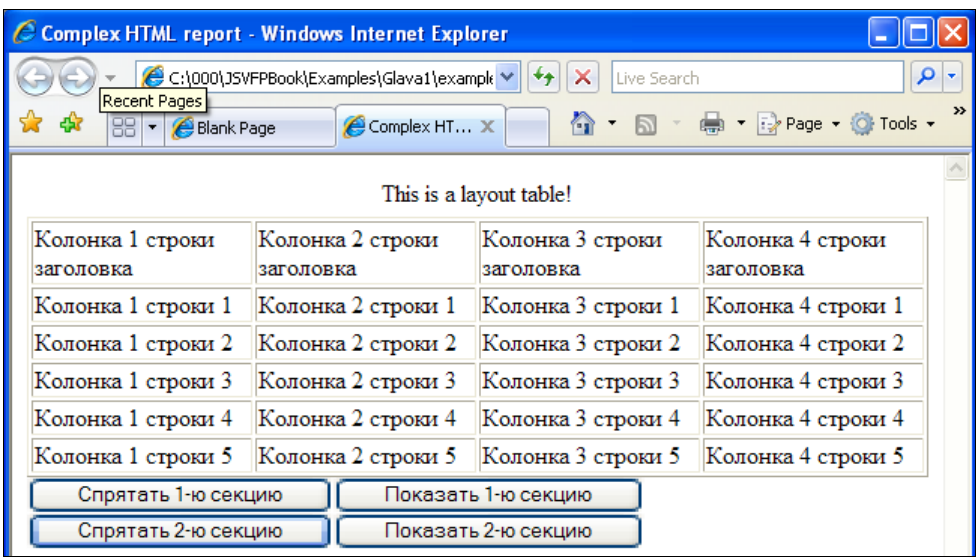


Рис. 1.3. Результат выполнения программы из листинга 1.4 (скрыта нижняя секция таблицы)

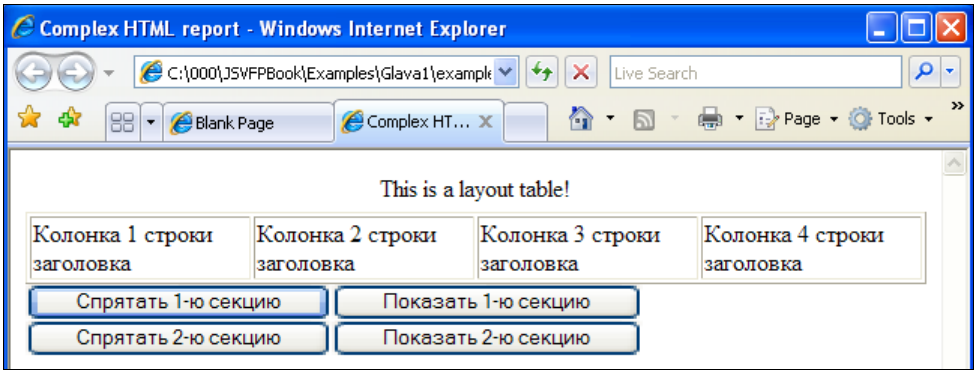


Рис. 1.4. Результат выполнения программы из листинга 1.4 (скрыты обе секции)

Практическое использование

Чем нам может быть полезен HTML/XHTML? Прежде всего, возможностью вывода в формах приложений документов Unicode. Но наиболее впечатляющее решение заключается в использовании этих языков для создания очень сложных отчетов.

Отчеты в формате HTML/XHTML

Предвижу недоумение читателя — к чему городить огород, когда мы можем вывести отчет в HTML-формате с помощью улучшенного генератора отчетов. Можем, однако добиться большой гибкости в раскладке, создание множества вложенных подотчетов любой глубины иерархии с помощью встроенного генератора отчетов Visual FoxPro очень непросто, а порою просто невозможно. С помощью HTML DOM это не просто "легко", а невероятно легко.

Кроме того, HTML DOM позволяет нам работать с элементами страниц, как с объектами, что подразумевает возможность работы с отчетами с использованием объектно-ориентированного программирования, а наличие достаточно широкого набора свойств, методов и событий позволяет создавать интересные и богатые по возможностям приложения. Я не оговорился, именно приложения, т. к. использование скриптов позволяет выводить на печать развернутые отчеты или отчеты со скрытыми частями. Звучит заманчиво! Сделать это не так сложно, как может показаться. Прежде всего, надо определиться с формулой раскладки комбинированного расчета. Использование служебных символов "\", "|", ":", "(" и ")" позволяет очень просто решить эту задачу. На-

пример, мне приходилось создавать комплексный отчет, раскладка которого выражена формулой:

"a / b | (c : d) (c : e) (c : f) (c) / g | h"

и означает, что отчет "a" располагается поверх всех отчетов, отчет "b" занимает левую треть нижележащего блока отчетов; для каждой строки отчета "c" имеются дополнительные отчеты "d", "e" и "f" и последняя строка отчета "c" несет суммарную информацию; отчеты "g" и "h" делят между собой нижнюю часть таблицы.

Структура такого отчета может быть представлена в виде таблицы раскладки, приведенной далее:

<table>
<tr><td colspan="2">Отчет "a"</td></tr>
<tr><td rowspan="7">Отчет "b"</td><td>Начало отчета "c"</td></tr>
<tr><td>Отчет "d", имеющий связанную информацию с началом отчета "c"</td></tr>
<tr><td>Продолжение отчета "c"</td></tr>
<tr><td>Отчет "e", имеющий связанную информацию с продолжением отчета "c"</td></tr>
<tr><td>Продолжение отчета "c"</td></tr>
<tr><td>Отчет "f", имеющий связанную информацию с продолжением отчета "c"</td></tr>
<tr><td>Окончание отчета "c"</td></tr>
<tr><td colspan="2">Отчет "g"</td><td>Отчет "h"</td></tr>
</table>

Возможно ли, быстро и просто создать подобный отчет, скомбинированный из 11 отдельных отчетов в стандартном генераторе отчетов Visual FoxPro? Если очень постараться, то в принципе возможно. А сделать его динамическим, позволяющим управлять печатью всего целиком, закрывать при печати отдельные части и при этом не вносить никаких изменений и условий в структуру отчета? Вряд ли. А с помощью объектов HTML DOM такой отчет был создан за несколько минут.

Разберем более простой вариант. Пусть наш отчет состоит из четырех отчетов и схема общего отчета соответствует формуле:

"Отчет A / Отчет B | Отчет C / Отчет D"

То есть нам нужна таблица раскладки 3×2 с тремя строками, с одной колонкой в первой строке, двумя колонками во второй строке и с одной колонкой в третьей. HTML-код такой таблицы представлен в листинге 1.5.

Листинг 1.5. HTML-код таблицы раскладки отчета

```
<table>
<thead>
<tr>
<th id="common_report_header" colspan="2">
Общий  заголовок для всех отчетов
</th>
</tr>
</thead>
<tbody>
<tr>
<td id="report_a" colspan="2">Заголовок и данные отчета A</td>
</tr>
<tr>
<td id="report_b">Заголовок и данные отчета B</td>
<td id="report_c">Заголовок и данные отчета C</td>
</tr>
<tr>
<td id="report_d" colspan="2">Заголовок и данные отчета D</td>
</tr>
</tbody>
</table>
```

Создавать таблицу с помощью объектной модели документа вы уже умеете. Теперь достаточно добавить элементам с идентификаторами `report_a`, `report_b`, `report_c` и `report_d` дочерние элементы, в качестве которых будут выступать HTML-таблицы.

Пример создания примитивного отчета подобной структуры приведен в проекте js_htmlreport.pjx, который вы найдете на прилагаемом к книге диске в каталоге JSVFPBook\Eamples\Glava1.

Некоторые нюансы использования HTML DOM с VFP

По рекомендации W3-консорциума атрибуты HTML-элементов должны быть заключены в двойные кавычки. Однако в случае создания атрибутов с помощью программного кода Visual FoxPro обрамления значений атрибута двойными кавычками не происходит, если значение атрибута не содержит пробелов. Тем не менее документ с таким содержанием будет корректно отображаться в разных интернет-обозревателях, но не сможет пройти проверку на анализаторе консорциума по адресу **http://validator.w3.org**.

Кроме того, при создании некоторых HTML-элементов, например элементов `meta`, атрибут *`name`* создается и принимает присваиваемое ему значение, однако при сохранении документа во внешний файл атрибут *`name`* теряется. Поэтому если предполагается сохранять создаваемые подобным образом файлы вне приложения, то для исправления коллекции атрибутов HTML-элементов `meta` придется использовать средства замены строки в уже готовом файле, например, использовать функцию `STRTRAN()` Visual FoxPro.

Код программы, демонстрирующий эти проблемы, приведен в листинге 1.6.

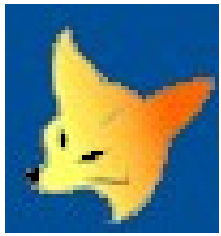
Листинг 1.6. Программа meta_challenges.prg

```
Clear
Local loHtmlFile As MSHTML.HTMLDocument, ;
   loHead As MSHTML.HTMLHeadElement, ;
   loMeta As MSHTML.IHTMLMetaElement,;
   lcInitialContent As String

loHtmlFile=Createobject("htmlfile")
lcInitialContent='<html><head></head><body></body></html>'
loHtmlFile.Write(lcInitialContent)
loHtmlDocument=loHtmlFile.documentElement
loHead=loHtmlFile.childNodes.Item(0).childNodes(0)

loMeta=loHtmlFile.createElement("meta")
loMeta=loHead.insertBefore(loMeta)
loMeta.Name="Author "
loMeta.Content="FoxPro programmer"
loMeta=loHtmlFile.createElement("meta")
loMeta.Name="Pragma"
loMeta.Content="no-cache "
loMeta=loHead.appendChild(loMeta)

loMeta=loHtmlFile.createElement("meta")
loMeta=loHead.insertBefore(loMeta)
loMeta.httpEquiv="Content-type"
loMeta.Content="text/html; charset=windows-1251"
loHtmlDocument=loHtmlFile.documentElement
* атрибут name отсутствует у meta-элементов
? loHtmlDocument.outerHtml
=Strtofile(;
   loHtmlDocument.outerHtml,;
   "C:\JSVFPBook\Examples\Glava1\meta_challenges.htm")
```

```
Local ;
   loIE As InternetExplorer.Application,;
   loDoc As MSHTML.HTMLDocument,;
   loMetaCollection As Object,;
   loMeta As Object

loIE=Createobject("InternetExplorer.Application")
loIE.Navigate(;
   "C:\JSVFPBook\Examples\Glava1\meta_challenges.htm";
   )
Do While loIE.ReadyState !=4
   DoEvents
Enddo
loDoc=loIE.Document
loMetaCollection=loDoc.getElementsByTagName("meta")
For Each loMeta In loMetaCollection
   ? loMeta.Name
   ? loMeta.Content
Endfor
Release All
```

Прекрасные примеры работы с объектами HTML-документов вы найдете в поставляемом с Visual FoxPro фундаментальном классе _html.vcx, размещенном в каталоге FFC домашнего каталога Visual FoxPro.

ГЛАВА 2



XML и другие X

Пожалуй, не найдется другого языка с такой же витиеватой историей, какая происходит по сей день с XML. Чего только не писалось о нем журналистами и околопрограммными авторами за десятилетие его развития. От невероятной по "глубине понимания предмета" фразы "SGML умер, да здравствует XML!", до сверхоптимистических надежд, вроде "XML — безальтернативное хранилище данных". Не вдаваясь в суть, да часто даже и не пытаясь понять, что же это такое, нас убеждали то в очередной революции, сметающей все на своем пути, то в очередном тупике, то в... — перечислять можно до бесконечности. Не отставал от журналистов и W3-консорциум. За неполных 10 лет консорциумом было издано четыре редакции одной и той же спецификации. А что же программисты? А программисты, следя за путями его развития, похожими скорее на запутанные схемы движения метро, гадали, куда же кривая выведет? Кривая вывела. Мы имеем стандарт, а также многочисленные его вольные трактовки. А что вы хотели от кривой?

Терминология

Погружение в изучение XML мне, лично, очень напоминало путешествие Алисы в страну чудес. "Сплошная путаница", — сказала бы она, почитав труды многих авторов. Так что же представляет собой XML?

Если исходить из его названия, то это сочетание одиночных символов из трех слов его определения — eXtensible Markup Language — расширяемый язык разметки.

Если рассматривать его физическую сущность, то XML, с одной стороны, может представлять собой обычный ASCII-файл, информация в котором оформлена определенным образом, с другой стороны — содержимое какого-

то поля базы данных. В первом случае мы говорим о XML-файле, во втором — о XML-документе.

В чем отличие XML от HTML? В первую очередь различие между этими двумя языками заключается в целях, ради которых эти языки были созданы. Если HTML изначально создавался для представления информации, то XML — для публикации больших массивов информации. Если HTML имеет фиксированный набор тэгов, составляющий словарь языка, то XML такого словаря не имеет, что позволяет разработчикам создавать свои собственные словари и использовать эти словари, когда требуется описать какие-то данные. Однако главное различие, и это различие является для нас наиболее важным, заключается в том, что HTML является языком *представления* данных, тогда как XML является языком *описания* данных.

Поскольку, как уже было сказано ранее, XML может быть представлен в виде физического, текстового ASCII-файла, то для нас крайне важен тот факт, что в таком случае можно говорить о кроссплатформенности XML. По своей природе XML является полностью интерпретируемым языком, а значит, может представлять и представляет собой средство обмена информации между приложениями, вне зависимости от того, на какой платформе эти приложения были созданы.

> ***Совет***
>
> Несмотря на сложность восприятия спецификации W3-консорциума из-за особенности изложения, именно она должна служить основой для понимания и изучения XML.

Well-formed против valid

Вероятно вы слышали такие определения XML-документов, как "well-formed" (хорошо сформированный) и "valid" (правильный). Многие уважаемые мной авторы иногда прямо делят XML-документы на указанные типы. Квинтэссенцией такого деления могут служить определения, приведенные в Википедии.

Хорошо сформированный (well-formed) XML-документ

"Хорошо сформированный XML документ" на **http://en.wikipedia.org/wiki/Well-formed_XML_document** определяют как XML-документ, который имеет корректный XML-синтаксис. В соответствии с рекомендацией W3-консорциума, это означает:

- XML-документы должны иметь корневой элемент;
- XML-элементы должны иметь закрывающий тэг;

- XML-тэги являются регистро-чувствительными;
- XML-элементы должны быть вложены должны образом;
- значения атрибутов XML должны быть всегда заключены в кавычки.

Далее следует утверждение, что не нужно путать этот тип с "правильным" XML-документом, который определен как "хорошо сформированный" XML-документ и который, кроме того, соответствует правилам определения типа документа (Document Type Definition (DTD)) или XML-схеме (XSD), которая поддерживается W3-консорциумом в качестве альтернативы определению типа документа.

Правильный (valid) XML-документ

Правильный XML-документ определен W3-консорциумом (**http://en.wikipedia.org/wiki/Valid_XML_document**), как "хорошо сформированный" документ, который, кроме того, соответствует правилам определения типа документа (Document Type Definition (DTD)) или XML-схеме (XSD), которая поддерживается W3-консорциумом в качестве альтернативы DTD.

Далее следует утверждение, что не нужно путать этот тип с "хорошо сформированным" XML-документом, который определен как имеющий корректный XML-синтаксис в соответствии со стандартами W3-консорциума.

Мне такие определения напоминают анекдот про две половинки таблетки от головы и живота — "и смотри, не перепутай!".

Что ясно и недвусмысленно сказано в спецификации W3-консорциума?

"Объект данных представляет собой XML-документ, если он хорошо сформирован так, как определено в этой спецификации. Дополнительно XML-документ является правильным, если он отвечает определенным дополнительным ограничениям".

Следовательно, "правильность" является не более чем свойством XML-документа, но уж никак не может быть отдельным типом. А плохо сформированный XML-документ вообще не является XML.

> ***Примечание***
>
> На самом деле, чтобы быть "хорошо сформированным" документ должен отвечать большему числу правил и ограничений. Мы разберем правила создания "хорошо сформированного документа" чуть позже, а пока будем считать приведенные ранее базовыми правилами.

Структура XML-документа

Несмотря на то, что структура документа, как логическая, так и физическая и их составляющие однозначно определены в спецификации W3-консорциума,