Введение в XML

ГЛАВА

21

В этой главе...

Расширяемый язык разметки (XML — Extensible Markup Language) представляет собой популярное средство представления данных. Несмотря на то что XML является урезанной версией стандартного обобщенного языка разметки (SGML — Standard Generalized Markup Language), именно он чаще всего встречается на практике. Язык XML применяется для хранения установок и данных приложений, поддерживает унифицированную структуру для передачи данных, инкапсулирует RSS-каналы на веб-сайтах и применяется для решения ряда других задач. Стандарты XML были адаптированы другими форматами данных, такими как HTML (путем разработки XHTML).

Эта глава представляет собой введение в XML и содержит описание формата данных, а также инструментов и методов, используемых для обработки информации.

Основы XML

В процессе создания языка XML преследовалась цель перенести преимущества стандарта SGML на меньшие по размеру платформы, такие как веб-браузеры. Язык XML сохранил гибкость своего предшественника, но при этом был существенно переделан, чтобы учесть особенности WWW. В результате значительно упрощена передача данных с помощью интернет-архитектуры и обеспечивается их отображение с меньшими издержками.

Стратегия XML-дизайна основана на следующих положениях.

- Форма должна соответствовать функциональности. Другими словами, язsык должен быть достаточно гибким, чтобы инкапсулировать многие типы данных. Вместо "впихивания" различных форм данных в одну структуру следует преобразовать саму структуру таким образом, чтобы она была адекватной данным.
- Документы должны распознаваться по их содержимому. Разметка должна проектироваться таким образом, чтобы не возникало никаких сомнений по поводу содержимого, которое она обрамляет. В случае выполнения этого требования XML-документ будет называться самоописывающим.

- Формат должен отделяться от представления. Язык разметки должен представлять различия между самими данными и не пытаться описывать представление самих данных. Например, элементы должны маркироваться тегами `<emphasis>`, а не `<b>` (bold). Благодаря этому представление данных (которые должны быть выделены, но необязательно полужирным шрифтом) делегируется платформе, на которой обрабатываются данные.
- Язык должен быть простым и легко поддаваться анализу при наличии внутренней проверки корректности кода.

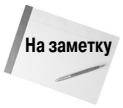
Ниже приведены рекомендации относительно языка XML, которые можно найти в документе консорциума W3C Recommendation for XML 1.0 (этот документ доступен по адресу `www.w3.org/TR/1998/REC-xml-19980210`):

- простота и очевидность применения любыми интернет-приложениями;
- поддержка широкого набора приложений;
- совместимость с языком SGML;
- упрощенное создание программ, предназначенных для обработки XML-документов;
- количество дополнительных свойств XML должно быть сведено к абсолютному минимуму, в идеальном случае — к нулю;
- простота распознавания человеком и отсутствие неоднозначности;
- минимизация времени, затраченного на разработку XML-проекта;
- формальность и лаконичность XML-проекта;
- простота создания;
- отсутствие избыточности для XML-разметки.

Вообще говоря, XML не слишком хорошо подходит для WWW. Это связано с тем, что элементы XML-документов определяются каждым разработчиком индивидуально, и браузеры не в состоянии интерпретировать и отображать все XML-документы в таком виде, как задумано автором. Но зато стандартизованные XML-структуры могут использоваться для хранения данных приложений. Для примера рассмотрим следующие применения XML.

- Популярный формат синдицирования RSS определяет теги элементов в XML-формате таким образом, чтобы инкапсулировать синдицированные новости и RSS-каналы. Благодаря этому многие приложения могут легко распространять информацию, которая содержится в RSS-канале.
- Многие сайты интерактивной статистики (например, сайты статистики компьютерных игр и т.д.) хранят информацию в XML-формате, благодаря чему она легко может анализироваться самыми разными приложениями.
- Многие приложения хранят свои настройки в XML-файлах. Эти файлы при необходимости можно легко проанализировать, изменить либо переписать.
- Многие текстовые процессоры и другие приложения, обрабатывающие документы (например, электронные таблицы), хранят документы в XML-формате.
- Многие приложения B2B используют XML для совместного использования данных и обмена ими.

Несмотря на то что XML поддерживает идеальную структуру данных для некоторых приложений, его лучше применять для обработки небольших последовательных коллекций данных. Если же используется случайная выборка данных или хранятся тысячи и десятки тысяч записей, лучше обратиться к базам данных.

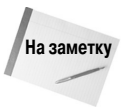


Язык XHTML разрабатывался, чтобы обеспечить совместимость языка HTML с XML (каждому открывающему тегу должен соответствовать закрывающий тег и т.д.). Иными словами, XHTML был приспособлен к стандартам XML, хотя сам по себе и не является расширяемым языком разметки.

Синтаксис языка XML

Требования к документам, созданным на языках XML и XHTML, во многом напоминают требования к HTML-документам, хотя и являются более строгими:

- имена элементов и атрибутов должны записываться с учетом чувствительности к регистру символов;
- все элементы должны быть корректно закрыты;
- элементы должны быть корректно вложенными и не перекрывать друг друга;
- все атрибуты должны иметь значения;
- все значения атрибутов должны заключаться в кавычки.



В этой книге рассматривается синтаксис форматирования XHTML, который является более формализованным, чем синтаксис HTML, и лучше соответствует соглашениям, принятым в XML.

Если структура документов подобна принятой в HTML, теги элементов могут применяться для инкапсуляции контента, который, в свою очередь, может включать контент, разделенный тегами.

В следующих разделах рассматривается синтаксис различных XML-элементов.

Начнем с пролога

В начале каждого XML-документа находится XML-декларация, которая выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Декларация начинается с ключевого слова `<?xml?>`, за которым следуют атрибуты `version` и `encoding`. С помощью атрибута `version` указывается версия XML, используемая в документе. Атрибут `encoding` определяет символьную кодировку, используемую при создании содержимого документа.

Как и в случае с другими языками разметки, XML поддерживает определения типа документа (DTD — Document Type Definition), с помощью которых задаются синтаксические правила для документов, содержащих DTD. Определения DTD используются в приложениях для проверки синтаксиса документов. Декларация XML-документа в виде определения DTD напоминает о том, что перед вами — XHTML-документ (для этого используется определение `SYSTEM` или `PUBLIC`). Например, следующее определение DTD используется для документов, созданных в OpenOffice:

```
<!DOCTYPE office:document-content PUBLIC
  "-//OpenOffice.org//DTD OfficeDocument 1.0//EN" "office.dtd">
```

Ниже приводится пример определения DTD для XHTML-документа:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Элементы

Элементы языка XML во многом напоминают HTML-элементы. Но поскольку язык XML является расширяемым, набор его элементов шире, чем набор HTML-элементов. Например, рассмотрим следующий фрагмент кода, заимствованный из RSS-канала, который представлен в XML-формате.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="/externalflash/NASA_Detail/NASA_Detail.xsl"
  type="text/xsl"?>
<rss version="2.0">
  <channel>
    <title>Последние новости от NASA</title>
    <link>http://www.nasa.gov/audience/formedia/features/index.html</link>
    <description>Лента новостей RSS, включающая последние новости от NASA,
      статьи и пресс-релизы.</description>
    <language>ru-ru</language>

    <item>
      <title>"Атлантис" возвращается в Космический центр имени
Кеннеди</title>
      <link>./HQ_M07077_Atlantis_ferry_flight.html</link>
      <description>Летающий паром на базе Боинга-747 с шаттлом
        "Атлантис" на борту прибывает в Космический центр имени Кеннеди
        в эти выходные.</description>
      <pubDate>Пятница, 29 июня 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>Отчет о состоянии ISS: SS07-32</title>
      <link>./HQ_SS0732_station_status.html</link>
      <description>Операции и исследования, выполненные нашими
        специалистами на этой неделе.</description>
      <pubDate>Пятница, 29 июня 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>Спутник сфотографировал первые серебристые облака </title>
      <link>./HQ_07145_AIM_First_Light.html</link>
      <description>Спутник NASA сфотографировал таинственные
        серебристые облака, которые впервые появились этим летом
        на высоте 80 км от поверхности Земли.</description>
      <pubDate>Четверг, 28 июня 2007 00:00:00 EDT</pubDate>
    </item>
    <item>
      <title>Марсоход NASA готов к спуску в кратер </title>
      <link>./HQ_07145_Rover_Victoria_Crater.html</link>
      <description>Марсоход NASA "Оппортьюнити" готов к спуску
        по каменистой насыпи в огромный кратер "Виктория", расположенный
        на Красной планете.</description>
      <pubDate>Четверг, 28 июня 2007 00:00:00 EDT</pubDate>
    </item>
</channel>
</rss>
```

В рассматриваемом примере используются следующие элементы.

- Элемент `channel`. Контейнер, в котором находится RSS-канал. В нем находятся следующие подконтейнеры:
  - `title` — название канала или фида;

- `link` — ссылка на канал в Интернете;
- `description` — описание канала;
- `language` — язык контента канала.

■ Элемент `item`. Каналы инкапсулируют каждую новость внутри элемента `item`, который может включать следующие подэлементы:

- `title` — заголовок новости;
- `link` — ссылка на новость в Интернете;
- `description` — краткое описание новости;
- `pubDate` — дата публикации новости.

Обратите внимание на то, что многие элементы могут включать разнородный контент. Например, элементы `channel` и `item` могут поддерживать контент для элементов `title` — выбор элементов, на которые ссылается элемент `title`, зависит от его размещения.

Атрибуты

Как и в XHTML, XML-элементы поддерживают атрибуты. Различие же заключается в том, что в XML разработчик может сам определять атрибуты в зависимости от назначения документа. Например, рассмотрим следующий фрагмент кода.

```
<employee sex="female">
  <lastName>Мур</lastName>
  <firstName>Терри</firstName>
  <hireDate>20-02-2003</hireDate>
</employee>
<employee sex="male">
  <lastName>Робинсон</lastName>
  <firstName>Брэндан</firstName>
  <hireDate>30-04-2000</hireDate>
</employee>
```

В рассматриваемом примере пол сотрудника кодируется как атрибут элемента `employee`.

В большинстве случаев вместо атрибутов можно использовать другие элементы. Например, в предыдущем примере пол сотрудника можно кодировать с помощью дочернего элемента, а не атрибута.

```
<employee>
  <sex>female</sex>
  <lastName>Мур</lastName>
  <firstName>Терри</firstName>
  <hireDate>20-02-2003</hireDate>
</employee>
<employee>
  <sex>male</sex>
  <lastName>Робинсон</lastName>
  <firstName>Брэндан</firstName>
  <hireDate>30-04-2000</hireDate>
</employee>
```

Способ кодирования данных зависит от того, каким образом интерпретируется контент элемента, — в качестве модификатора или данных. Если приложение будет использовать контент в качестве данных, лучше кодировать его внутри элемента, поскольку это облегчит синтаксический анализ в дальнейшем.

Комментарии

Тег комментария в XML аналогичен тегу комментария в HTML.

```
<!-- комментарий /-->
```

Комментарии можно включать в любое место XML-документа, если при этом не нарушаются стандартные соглашения XML и соответствующие правила, заданные в DTD.

Исключение данных из процесса анализа

Иногда приходится определять контент, который не должен анализироваться (интерпретироваться приложением, которое считывает данные). Для этого определяются символьные данные. Не анализируемые данные определяются с помощью элемента CDATA, как показано ниже.

```
<!CDATA [исключение данных из процесса анализа]]>
```

Обычно элементы CDATA применяются для улучшения читаемости документов путем включения зарезервированных символов в область действия элемента CDATA вместо использования замысловатых сущностей. Например, оба приведенных ниже элемента форматирования абзаца возвращают одни и те же данные, но первый из них лучше читается, поскольку вместо сущностей используются элементы CDATA.

```
Элемент table должен использоваться вместо элемента pre,
если это возможно.
Элемент <!CDATA [table]]> должен использоваться вместо элемента <!CDATA
[pre]]>, если это возможно.
```

Сущности

В XML можно определять пользовательские сущности, которые представляют собой мнемонические коды, определяющие некоторое содержимое. Для определения сущностей применяется следующий синтаксис:

```
<!ENTITY имя_сущности "значение_сущности">
```

Сущности задаются в определении DTD документа. Например, в следующей декларации XML-документа в качестве значения сущности customer определяется "Acme, Inc".

```
<?xml version="1.0"?>
<!DOCTYPE report SYSTEM "/xml/dtds/reports.dtd" [
  <!ENTITY customer "Acme, Inc.">
]>
```

Затем сущность можно включить в любое место документа (с помощью инструкции &*имя_сущности*;).

```
<report>
  <title>Отчет о тестировании</title>
  <date>25-01-2005</date>
  <summary>В результате выполнения последнего регрессионного тестирования
  были получены превосходные результаты. Теперь заказ для
  &customer; завершен, и можно предоставить финальный код.</summary>
...
```

Сущности можно объявлять как внешние ресурсы. Подобные внешние ресурсы обычно занимают гораздо больше места, чем несколько слов или фраза, и представляют собой завершенные документы. Для определения системной сущности, применяемой для декларирования внешних ресурсов, используется следующий синтаксис:

```
<!ENTITY имя_сущности SYSTEM "URL">
```

Например, следующий код определяет сущность `chapter01`, которая ссылается на локальный документ `chapter01.xml`:

```
<!ENTITY chapter01 SYSTEM "chapter01.xml">
```

Сущность `chapter01` может применяться для включения содержимого документа `chapter01.xml` в текущий документ.

Пространства имен

Концепция пространств имен начала использоваться в XML относительно недавно. Благодаря пространствам имен можно группировать элементы в соответствии с их назначением, используя уникальное имя. Подобная операция может преследовать самые разные цели, но чаще всего она применяется для различения элементов.

Например, элемент `table` может иметь отношение как к элементу данных, так и к физическому объекту, например к столу, находящемуся на кухне.

```
<!-- Элемент данных в одном документе -->
<table>
  <tr><th>Date</th><th>Заказчик</th><th>Количество</th></tr>
  <tr><td>25-01-2005</td><td>Acme, Inc</td><td>125.61</td></tr>
...
</table>

<!-- Домашняя мебель в другом документе /-->
<table>
  <type>Обеденный</type>
  <width>4</width>
  <length>8</width>
  <color>Вишня</color>
</table>
```

Если оба элемента используются в одном и том же документе, они могут конфликтовать, поскольку обозначают два совершенно разных предмета. Конфликта можно избежать, если использовать пространство имен. Для обозначения пространства имен к именам элементов добавляются соответствующие префиксы. Например, для обозначения элементов `table`, имеющих отношение к мебели, используется пространство имен `furniture`.

```
<furniture:table>
  <type>Обеденный</type>
  <width>4</width>
  <length>8</width>
  <color>Вишня</color>
</furniture:table>
```

Префикс должен быть уникальным образом связан с декларацией пространства имен с помощью атрибута `xmlns`. Декларация пространства имен записывается в следующем виде:

```
<prefix:tag xmlns:tag="url">
```

Например, с помощью префикса `furniture`, предваряющего тег `table`, можно составить следующую инструкцию:

```
<furniture:table xmlns:table="http://www.w3.org/XML/">
```

Обратите внимание на то, что URL-ссылка в декларации используется лишь в качестве уникального идентификатора и не воспринимается XML-анализатором в качестве единого указателя ресурсов какого-либо типа и может даже не существовать. Но в любом случае она должна быть уникальной в области своего действия.

Таблицы стилей

В XML также осуществляется поддержка таблиц стилей, которые связываются с XML-документами с помощью тега `xml-stylesheet`, имеющего следующий синтаксис:

```
<?xml-stylesheet type="тип MIME" href=
"url-ссылка на таблицу стилей"?>
```

Например, для ссылки на таблицу стилей в XML-документе следует использовать следующий тег:

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

Работа с определениями типа документа

Как упоминалось ранее, если XML-документ соответствует синтаксическим правилам, принятым в XML, он будет называться *правильно скомпонованным документом*. Документ также может быть действительным или недействительным. Документ называется *действительным*, если проходит проверку по отношению к правилам, определенным в DTD. Как и в случае с HTML, XML DTD представляет собой документ, содержащий набор правил, которые определяют структуру XML-документа. Например, DTD может включать правило, которое определяет, что все элементы контактов должны включать элемент `phone`, как указано в следующем примере кода.

```
<contact>
  <name>Джилл Хеннесси</name>
  <address>111 Ист-Мэйн Стрит</address>
  <phone>1-303-555-4444</phone>
</contact>
```

Предыдущий фрагмент кода является правильно скомпонованным, как и следовало ожидать. Также можно дополнительно определить правила, которые более четко определяют назначение каждого элемента, а также его положение в структуре документа. Можно также определить, сколько раз каждый элемент может (или должен) появляться в документе.

Определение DTD может находиться вне XML-документа, действительность которого оно определяет, либо внутри его. Если определение DTD находится вне документа, следует его объявить внутри XML-документа, благодаря чему XML-анализатор получит ссылку на местонахождение внешнего определения DTD:

```
<!DOCTYPE root SYSTEM "имя_файла">
```

Например, в XML-документе, определяющем контактные данные, внешняя декларация `DOCTYPE` может выглядеть так, как указано в следующем примере кода (здесь декларация `DOCTYPE` выделена полужирным шрифтом).

```
<?xml version="1.0"?>
<!DOCTYPE contact SYSTEM "contact.dtd">
<contact>
  <name>Джилл Хеннесси</name>
  <address>111 Ист-Мэйн Стрит.</address>
  <phone>1-303-555-4444</phone>
</contact>
```

Определения DTD должны помещаться в отдельный файл, `contact.dtd`, доступ к которому осуществляется из исходного XML-документа.

Правила `DOCTYPE` можно также поместить в сам XML-документ, как показано в следующем примере.

```
<?xml version="1.0"?>
<!DOCTYPE contact [
  <!ELEMENT contact (name, address, phone)>
    <!ELEMENT name     (#PCDATA)>
    <!ELEMENT address  (#PCDATA)>
    <!ELEMENT phone    (#PCDATA)>
]>
<contact>
  <name>Джилл Хеннесси</name>
  <address>111 Ист-Мэйн Стрит.</address>
  <phone>1-303-555-4444</phone>
</contact>
```

Хотя определения DTD не являются XML-документами, структура определения DTD и XML-документа одна и та же, включая следующие основные компоненты XML:

- элементы;
- атрибуты;
- сущности;
- разделы PCDATA;
- разделы CDATA.

Каждый из этих компонентов будет описан далее.

Использование элементов в определениях DTD

Элементы — это основные информационные компоненты XML. Они применяются для формирования структуры документа. Ранее вы встречались с ними в многочисленных примерах HTML-кода, и главные принципы их использования в XML-коде остались прежними. Элементы могут включать данные либо быть пустыми. Если элемент пустой, он обычно включает атрибут, хотя и необязательно. HTML-элементы `br` и `img` являются хорошими примерами пустых элементов, не включающих каких-либо данных.

Элементы XML определяются с помощью декларации `element` и следующего синтаксиса:

```
<!ELEMENT name тип_данных>
```

Первая часть декларации (`!ELEMENT`) отвечает за определение элемента. Следующая часть (`name`) отвечает за определение имени элемента. Последняя часть декларации (*тип_данных*) объявляет тип данных, которые может содержать элемент. Обычно элемент может содержать следующие типы данных, заданные с помощью определений DTD:

- данные EMPTY, которые означают, что элемент не содержит данных;
- данные PCDATA или проанализированные символьные данные;
- один или несколько дочерних элементов.

Синтаксис объявления элементов для пустых элементов

Для объявления пустых элементов применяется ключевое слово EMPTY.

```
<!ELEMENT name EMPTY>
```

Например, для объявления пустого элемента `rug` необходимо воспользоваться следующим оператором:

```
<!ELEMENT rug EMPTY>
```

Этот элемент будет выглядеть следующим образом в составе XML-документа:

```
<rug />
```

Синтаксис определения элементов с помощью структуры PCDATA

Элементы, которые не содержат дочерних элементов и включают лишь символьные данные, определяются с помощью ключевого слова #PCDATA, заключенного в скобки, как показано в следующем примере:

```
<!ELEMENT name(#PCDATA)>
```

Ниже приведен типичный пример записи этого элемента:

```
<!ELEMENT note (#PCDATA)>
```

После определения элемента note XML-анализатор может найти его в XML-коде.

```
<note>Сауны могут быть закрыты в связи с техническим обслуживанием
всю следующую неделю. Пожалуйста, поставьте своих клиентов
 в известность.</note>
```

Как показано в предыдущем примере, элемент note содержит текст (определен в разделе PCDATA) и не включает дочерних элементов.

Синтаксис для определения элементов, имеющих дочерние элементы

Элементы могут включать последовательности, состоящие из одного или нескольких дочерних элементов, которые определяются с помощью имен дочерних элементов, указанных в скобках.

```
<!ELEMENT name (имя_дочернего_элемента)>
```

Если имеется несколько дочерних элементов, они будут разделены запятыми:

```
<!ELEMENT name (имя_дочернего_элемента_1,
имя_дочернего_элемента_2)>
```

В рассмотренном ранее примере кода, содержащем информацию о контактах, элементы, имеющие дочерние элементы, определяются следующим образом:

```
<!ELEMENT contact (name, address, phone)>
```

Определение количества вхождений элементов

С помощью *оператора вхождения*, включаемого в декларацию элемента, можно определить частоту отображения одного элемента внутри другого. Знак “плюс” (+) показывает, что элемент должен встречаться один или более раз внутри другого элемента. Например, следующая декларация определяет, что элемент phone должен встретиться как минимум один раз внутри элемента contact:

```
<!ELEMENT contact (phone+)>
```

Можно объявить группу элементов, которая должна отобразиться один или более раз:

```
<!ELEMENT contact (name, address, phone)+>
```

Если элемент не будет встречаться либо будет встречаться заданное число раз (другими словами, задан необязательный элемент), вместо знака “плюс” воспользуйтесь знаком “звездочка”, как показано ниже:

```
<!ELEMENT contact (phone*)>
```

Если элемент не будет встречаться либо будет встречаться один раз (не будет встречаться более одного раза), следует воспользоваться знаком вопроса (?):

```
<!ELEMENT contact (phone?)>
```

Следующий XML-код будет недействительным, если в декларации для элемента phone используется оператор ?:

```
<contact>
  <phone>303-555-4444</phone>
  <phone>303-555-4447</phone>
</contact>
```

Если нужно указать, что один элемент может находиться внутри другого элемента, можно использовать вертикальную черту (|):

```
<!ELEMENT contact (name,address,phone,(email | fax))>
```

В предыдущей декларации последовательность элементов name, address и phone должна отображаться в указанном порядке, причем за ними должны следовать элементы email и fax, т.е. следующий XML-код будет действительным.

```
<contact>
  <name>Джилл Хеннесси</name>
  <address>111 Ист-Мэйн Стрит.</address>
  <phone>1-303-555-4444</phone>
  <email>jill@oasisoftranquility.com</email>
</contact>
```

Но следующий XML-код будет недействительным, если выполнить проверку на действительность по отношению к тому же самому определению DTD.

```
<contact>
  <name>Джилл Хеннесси</name>
  <address>111 Ист-Мэйн Стрит.</address>
  <phone>1-303-555-4444</phone>
  <email>jill@oasisoftranquility.com</email>
  <fax>303-555-4447</fax>
</contact>
```

Использование атрибутов в определениях DTD

Атрибуты определяют свойства элемента. Например, в HTML элемент img имеет свойство (или атрибут) src, которое описывает местонахождение изображения.

Для определения атрибутов элементов используется декларация ATTLIST. Эта декларация имеет следующий формат:

```
<!ATTLIST имя_элемента имя_атрибута
тип_атрибута значение_по_умолчанию>
```

Параметры *имя_элемента* и *имя_атрибута* играют роль, о которой свидетельствует их название (определяют элемент, к которому применяется атрибут, и имя фактического атрибута соответственно). Параметры *тип_атрибута* и *значение_по_умолчанию* являются более сложными, поскольку они должны обрабатывать несколько значений.

В табл. 21.1 приведены значения параметра *тип_атрибута*.

Таблица 21.1. Тип атрибута

Значение	Определение
CDATA	Символьные данные
(*значение*|*значение*|...)	Нумерованные данные
ID	Уникальный идентификатор

Окончание табл. 21.1

Значение	Определение
IDREF	Идентификатор другого элемента
IDREFS	Перечень идентификаторов других элементов
NMTOKEN	XML-имя
ENTITY	Сущность
ENTITIES	Перечень сущностей
NOTATION	Имя нотации
xml:	Предопределенное значение

В табл. 21.2 приводится перечень допустимых значений атрибута *значение_по_умолчанию*.

Таблица 21.2. Значения атрибута, заданные по умолчанию

Значение	Определение
значение	Атрибуту присвоено значение, заданное по умолчанию
#REQUIRED	Атрибуту элемента всегда должно присваиваться значение
#IMPLIED	Атрибут необязательно должен включаться в элемент
значение #FIXED	Атрибуту присвоено заданное по умолчанию значение, которое является фиксированным, т.е. не может изменяться автором

Следующая DTD-декларация определяет элемент phonenumber с атрибутом, заданным по умолчанию (home).

```
<!ELEMENT phonenumber (#PCDATA)>
  <!ATTLIST phonenumber type CDATA "home">
```

Чтобы ограничить значения атрибута type вариантами home (по умолчанию), work, cell или fax, декларацию необходимо изменить следующим образом:

```
<!ATTLIST phonenumber type (home|work|cell|fax)>
```

Использование сущностей в определениях DTD

Порядок создания сущностей в XML-документах рассматривался в разделе “Сущности” . Сейчас же позвольте напомнить синтаксис декларации ENTITY.

```
<!ENTITY имя_сущности "значение_сущности">
```

Сущности задаются в определении DTD XML-документа. Например, в следующей XML-декларации в качестве сущности customer определяется "Acme, Inc.":

```
<?xml version="1.0"?>
<!DOCTYPE report SYSTEM "/xml/dtds/reports.dtd" [
  <!ENTITY customer "Acme, Inc.">
]>
```

Теперь для включения в любом месте документа имени заказчика можно воспользоваться определенной выше сущностью (путем применения оператора &customer;).

Использование разделов PCDATA и CDATA в определениях DTD

Название PCDATA произошло от словосочетания "parsed character data" ("проанализированные символьные данные"), оно означает, что все символьные данные проанализированы XML-анализатором, распознаны все открывающие и закрывающие теги и раскрыты сущности. Раздел PCDATA могут содержать элементы.

В разделе CDATA находятся данные, которые не проанализированы XML-анализатором. Это означает, что теги не распознаны, а сущности не раскрыты. Обычно этот раздел содержат атрибуты, которые не могут содержать раздел PCDATA.

Введение в XML-схемы

Несмотря на всю свою полезность, определения DTD ограничены. Например, рассмотрим следующий XML-документ.

```
<datatypes>
  <Boolean>true</Boolean>
  <integer>1</integer>
  <double>563.34</double>
  <date>06-01-2007</date>
</datatypes>
```

Правила, заданные в определении DTD, требуют, чтобы каждый элемент включал только символьные данные. В результате значение элемента integer не является целочисленным, а значение элемента date не является датой. Причина подобного явления состоит в том, что определение DTD не включает математические и булевы типы данных и даты.

Консорциум W3C предложил разработчикам другую методологию разработки правил, которая получила название "XML-схема" (XML Schema). Эта методология предусматривает возможность применения большего количества типов данных и детализированный набор правил, которые позволяют добиться большего уровня спецификации (по сравнению с определениями DTD). Помимо управления типами правил DTD, схемы обеспечивают управление типами данных, разрешенными для элемента, например булевыми или целочисленными.

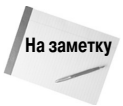
Применение различных типов данных приобретает особую важность в связи с тем, что они облегчают работу с традиционными базами данных и интерфейсами прикладного программировани (API), основанными на Java, C++ и других языках программирования, таких как JavaScript.

Работа со схемами

Теперь, когда вы уже познакомились с определениями DTD, не составляет особого труда расширить эти концепции на более широкий диапазон типов данных. В XML-схемах используется синтаксис языка XML для разработки набора правил, который является немного более интуитивно понятным, чем рассмотренный ранее синтаксис определений DTD.

Вспомните приведенный ранее пример создания простого XML-документа, содержащего сведения о контактах, который был создан на основе определения contact.dtd. Приведенный ниже листинг кода иллюстрирует аналогичные принципы, которые применяются для работы со схемами. Особое внимание уделите потомкам элемента xs:sequence xs:element (выделены полужирным шрифтом), которые относятся к элементу xs:complexType.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.tumeric.net/schemas"
xmlns="http://www.tumeric.net/schemas"
elementFormDefault="qualified">
<xs:element name="contact">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="postalcode" type="xs:string"/>
      <xs:element name="age" type="xs:integer" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```



Элемент `contact` представляет собой сложный тип элемента, поскольку содержит другие элементы. Если же элемент не содержит дочерних элементов, он относится к простому типу.

В определении DTD последовательность элементов, которые будут отображаться в XML-документах, определяется путем заключения разделенного запятыми списка в определение элемента. В XML-схеме последовательность определяется путем расположения элементов в нужном порядке с помощью элементов `xs:sequence`. По сути она является частью большего определения корневого элемента XML-документа, в качестве которого используется элемент `contact`. Обратите внимание на использование атрибута `type` для элементов `xs:element`, который определяет тип данных для каждого элемента.

С помощью XML-схем для XML-элементов становятся доступными числовые типы данных. Если вы знакомы с языком программирования Java, то сможете легко заметить, что большинство типов данных, определенных в XML-схеме, подобны типам данных Java. Если же вы не знакомы с Java, то обратите внимание на следующие четыре базовых типа данных, используемых в XML-схемах:

- числовые (целочисленные и с плавающей запятой двойной точности);
- дата;
- строка;
- булевы.

Сведения о типах данных, используемых в XML-схемах, можно найти по такому адресу:
`www.w3.org/TR/xmlschema-2`

XML-схему можно поместить во внешний документ, а затем можно добавить ссылку на нее в свой XML-документ. Эта ссылка имеет следующий синтаксис.

```
<?xml version="1.0"?>
<contact xmlns="http://www.tumeric.net/schemas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.on-target-games.com/schemas/contact.xsd" >
  <name>Джонни Рудь</name>
```

```
  <address>111 Ист Онион Авеню.</address>
  <city>Биг Сити</city>
  <state>Калифорния</state>
  <postalcode>96777</postalcode>
  <phone>1-323-456-4444</phone>
  <fax>тест</fax>
  <email>rude@rude.com</email>
</contact>
```

Ссылка на схему определяется с помощью пространства имен в документе. Для объявления пространства имен может использоваться следующий синтаксис.

```
xmlns="http://www.tumeric.net/schemas"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.on-target-games.com/schemas/contact.xsd"
```

Дополнительные две строки кода включают добавочные пространства имен, которые применяются в качестве идентификаторов. Они передают анализатору сведения о том, что связанные с ними элементы являются уникальными и могут иметь специальным образом разработанные определения. Важной частью пространства имен является унифицированный идентификатор ресурса (URI — Uniform Resource Identifier), благодаря которому пространство имен становится уникальным. Если элементы находятся в заданном пространстве имен, которое управляется определенной схемой, они должны подчиняться правилам этой схемы.

Первое пространство имен в предыдущем фрагменте кода ссылается на пространство имен, заданное в схеме, которое уникальным образом связывает схему с указанным ресурсом (в рассматриваемом случае — с веб-сайтом). На самом деле в данном случае не устанавливается ссылка на веб-сайт, да и сама ссылка не является физическим указателем. Использование URI-ссылки обеспечивает простой способ установки идентичности, поскольку веб-сайт должен быть уникальным. Если же уникальность не гарантируется в силу того, что любой пользователь может воспользоваться адресом вашего веб-сайта в собственной схеме, вместо имени веб-сайта лучше использовать длинный набор символов, подобный указанному ниже:

```
xmlns="hk45kskds-scld456ksaldkttsslae697hg"
```

Второе пространство имен ссылается на местоположение схемы W3C, поэтому XML-процессоры могут проверять действительность XML-документа по заданной схеме. Эта процедура обязательна, поскольку вам придется вызывать ресурс, который вы используете (в данном случае — схему, которая может быть найдена с помощью пути, указанного с помощью атрибута `xsi:SchemaLocation`). Как только процессор находит схему, он пытается проверить действительность XML-документа в момент загрузки. Если XML-документ не соответствует правилам, установленным в определении схемы, появляется сообщение об ошибке (при этом предполагается, что используемый вами синтаксический анализатор может работать с XML-схемами).

Использование XML

Для фактического использования XML при создании документов требуется их предварительное преобразование в удобный формат. Обратите внимание, что количество средств и форматов, предназначенных для работы с XML, ограничено лишь вашей фантазией.

Для просмотра XML-документов не нужны специальные инструменты. Многие из современных браузеров включают средства, обеспечивающие просмотр XML-документов и реализацию целого ряда дополнительных возможностей, таких как выделение тегов и свертывание разделов документа. На рис. 21.1 показано окно браузера Internet Explorer с RSS-документом.

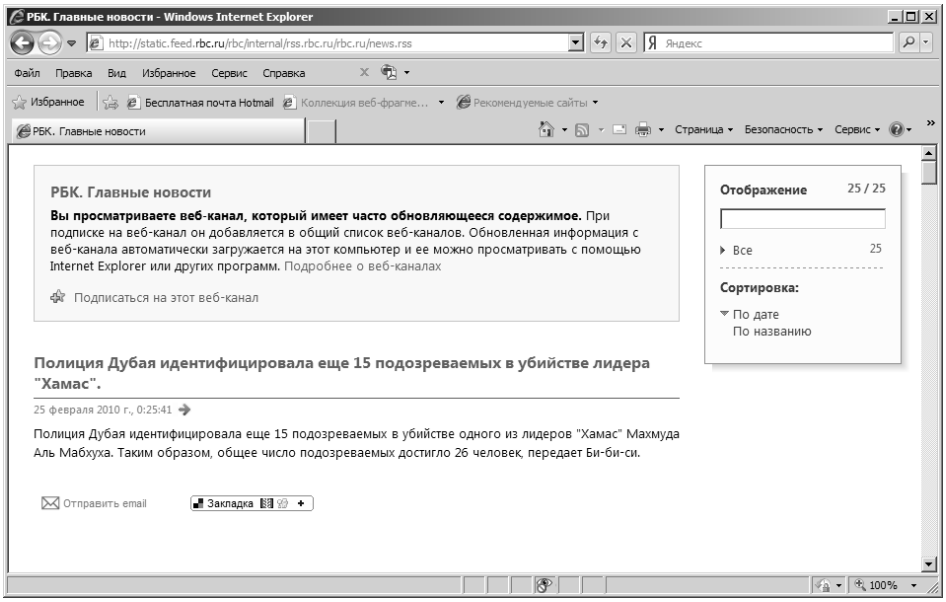


Рис. 21.1. XML-документ, отображаемый в окне браузера Internet Explorer

Преобразования XSLT

Расширяемый язык преобразования стилей (XSLT — Extensible Stylesheet Language Transformations) позволяет преобразовывать исходные XML-документы в отформатированные документы. В процессе преобразования содержимого документов изменяется расположение элементов исходного документа. При этом могут генерироваться новые элементы документа. В качестве входных данных XSLT используются XML-документ (иногда называемый *исходным деревом*) и таблица стилей, с помощью которой определяется сама трансформация. Выходной документ (который иногда называется *результирующим деревом*) приобретет нужный формат и будет готов к выводу на выбранном пользователем устройстве вывода.

Для управления XML-документами и выполнения преобразований XSLT доступен целый ряд инструментов, включая решения с открытым кодом (для получения доступа к таким инструментам просто введите “XSLT” в строке поиска на сайте `www.sourceforge.org`).

Редактирование XML-кода

Редактировать XML-файлы можно по-разному. Поскольку XML-формат по сути является текстовым форматом, для создания и редактирования XML-документов можно воспользоваться любым текстовым редактором (Emacs, `vi`, Блокнот и т.д.). Но все же лучше обратиться к специализированным XML-редакторам, которые значительно облегчат работу благодаря различным дополнительным возможностям — выделение и проверка синтаксиса, проверка действительности XML-документа, автозавершение кода и др. Ниже перечислены доступные в настоящее время XML-редакторы.

- Множество XML-редакторов с открытым кодом можно найти на сайте `http://sourceforge.org` (просто введите в строке поиска фразу “XML editor”).

- Леннарт Стафлин разработал XML-редактор из категории редакторов Emacs, получивший название "PSGML" (`www.lysator.liu.se/projects/about psgml.html`).
- Редактор XMetal, разработанный компанией Corel и позднее приобретенный компанией Blast Radius. Это хорошо известный, обладающий приличным набором возможностей XML-редактор (`www.xmetal.com`). Он распространяется на коммерческой основе и стоит весьма недешево.
- Редактор XMLSpy от фирмы Altova — еще один мощный XML-редактор, который продается по цене, сопоставимой с ценами на редактор XMetal, хотя персональные версии XML-редактора XMLSpy распространяются бесплатно (`www.altova.com`).
- Редактор <oXygen/> от SyncRO Soft — распространяемый по невысокой цене мультиплатформенный XML-редактор и XSLT-отладчик (`www.oxygenxml.com`).

Синтаксический анализ XML-кода

В настоящее время доступны приложения, предназначенные для синтаксического анализа XML-документов. В их число входят приложения с открытым кодом (для их поиска введите "XML parsing" в строке поиска на сайте `http://sourceforge.org`). Также модули синтаксического анализа XML-кода встроены во многие языки программирования.

- Джеймс Кларк предложил пользователям XML-документов анализатор XML-кода `expat`, который стал стандартом в мире анализаторов XML-кода (`http://expat.sourceforge.net` и `www.jclark.com/xml/expat.html`).
- В сети CPAN доступны XML-модули для Perl (`www.cpan.org`).
- Предлагается ряд XML-инструментов для платформы Python, включая те из них, которые перечислены на веб-сайте Python (`http://pyxml.sourceforge.net/topics`).
- В PHP имеются XML-функции, которые встроены в качестве расширений `expat` (`www.php.net/manual/en/ref.xml.php`).
- В хранилище PHP Extension and Application Repository находятся дополнительные расширения, обеспечивающие поддержку и обработку XML-документов (`http://pear.php.net`).

Резюме

В этой главе были рассмотрены основы XML — надежного расширяемого языка разметки, который может применяться для представления разнообразных данных. Вы убедились в том, что XML и HTML подобны по структуре, но при этом особенности XML позволяют этому языку успешно выполнять свои собственные функции. Здесь изучались такие вопросы, как определение XML-документов, объявление его элементов и способы расширения возможностей языка.

В оставшихся главах этой части рассматриваются язык XHTML Basic, методы очистки и проверки действительности документов, а также полезные приемы, используемые при разработке HTML-документов.