Document Object Model

DOM

DOM (от англ. *Document Object Model* — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам по лучить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

DOM

Модель DOM не налагает ограничений на структуру документа. Любой документ известной структуры с помощью DOM может быть представлен в виде дерева узлов, каждый узел которого представляет собой элемент, атрибут, текстовый, графический или любой другой объект. Узлы связаны между собой отношениями "родительский-дочерний".

Изначально различные браузеры имели собственные модели документов (DOM), несовместимые с остальными. Для того чтобы обеспечить взаимную и обратную совместимость, специалисты международного консорциума W3C классифицировали эту модель по уровням, для каждого из которых была создана своя спецификация. Все эти спецификации объединены в общую группу, носящую название W3C DOM.

Традиционный DOM

JavaScript был выпущен Netscape Communications в 1996 году в рамках Netscape Navigator 2.0. Конкурент Netscape – Microsoft выпустил позже в том же году Internet Explorer 3.0 с портом JavaScript, названным JScript. JavaScript и JScript позволяют разработчикам создавать интерактивные веб-страницы со стороны клиента. Ограниченные возможности обнаружения созданных пользователем событий и изменение документа HTML в первом поколении этих языков в итоге стали известны как “DOM уровень 0” или “Традиционный DOM”. Ни один независимый стандарт не был разработан для DOM уровень 0, но он был частично описан в спецификации HTML4.

Традиционный DOM

Традиционный DOM был ограничен в типах элементов, к которым можно получить доступ. Такие элементы, как форма (form), ссылка (link) и изображение (image) могли быть ссылками с иерархическими именами, которые начинались с корня объекта документа. Иерархическое имя могло использовать либо имена, либо последовательный индекс общего элемента. Например, элемент form input может быть доступен как "document.formName.inputName" или как "document.forms[0].elements[0]".

Традиционный DOM давал возможность подтверждения формы с клиентской стороны и популярный эффект «трансформации объекта».

Промежуточный DOM

В 1997 году Netscape и Microsoft выпустили Netscape Navigator и Internet Explorer версии 4.0, добавив поддержку Dynamic HTML (DHTML), предоставляющего возможность изменения функциональности HTML документа при его загрузке. DHTML требовал расширений для элементарного объекта document, который был доступен в Традиционной реализации DOM. Хотя Традиционная реализация DOM была в значительной степени совместимой с того момента, как JScript был основан на JavaScript, расширения DOM для DHTML были разработаны параллельно каждым из создателей браузера и остались несовместимыми. Эти версии DOM стали известны как «Промежуточный DOM».

Промежуточный DOM

Промежуточные DOM давали возможность манипуляции свойствами Cascading Style Sheets (CSS), которые воздействуют на отображение документа. Они также обеспечивают доступ к новому свойству под названием «слои» через свойства «document.layers» (в Netscape Navigator) и «document.all» (в Internet Explorer). Из-за первоначальной несовместимости в Промежуточных DOM разработка общего браузера потребовала специальной обработки для каждого случая.

Более поздние версии Netscape Navigator отказались от поддержки Промежуточного DOM. Internet Explorer продолжает поддержку своего Промежуточного DOM для обратной совместимости.

Стандартизация

Организация World Wide Web Consortium (W3C), основанная в 1994 году, чтобы развивать и поддерживать открытые стандарты для World Wide Web, заставила Netscape Communications и Microsoft вместе с другими компаниями разработать стандарт для скриптовых языков браузера под названием «ECMAScript». Первая версия стандарта была опубликована в 1997 году. Более поздние выпуски JavaScript и JScript будут осуществлять стандарт ECMAScript для большей межбраузерной совместимости.

После выхода ECMAScript W3C начала работу над стандартизацией DOM. Изначальный стандарт DOM, также известный как «DOM уровень 1», был рекомендован W3C в конце 1998 года. Примерно в это же время вышел Internet Explorer 5.0 с ограниченной поддержкой DOM уровень 1. DOM уровень 1 обеспечил полную модель для всего HTML- или XML-документа, включая способ изменения любой части документа. Неадаптированные браузеры, например, Internet Explorer 4.x и Netscape 4.x, были широко используемы вплоть до 2000 года.

Стандартизация

DOM уровень 2 был опубликован в конце 2000 года. Он ввел функцию "getElementById", а также модель событий и поддержку XML namespace и CSS. DOM уровень 3 — текущая версия спецификации DOM, опубликованная в апреле 2004 года, добавила поддержку XPath и обработку событий клавиатуры, а также интерфейс для сериализации документа как XML.

В 2005 году большая часть W3C DOM поддерживалась основными, удовлетворяющими ECMAScript, браузерами, включая Microsoft Internet Explorer version 6(2001 год), Opera, Safari и браузеры, основанные на Gecko (такие как Mozilla, Firefox, SeaMonkey и Camino).

Обработка XML-данных с использованием модели DOM

Модель DOM рассматривает XML-данные как стандартный набор объектов и используется для обработки XML-данных в памяти. Пространство имен **System.Xml** обеспечивает программное представление XML-документов, фрагментов, узлов и наборов узлов. Оно основывается на рекомендациях базовой модели DOM уровня 1 и модели DOM уровня 2 консорциума W3C.

Обработка XML-данных с использованием модели DOM

Класс XmlDocument представляет XML-документ. Он включает элементы для получения и создания всех других XML-объектов. С помощью класса XmlDocument и связанных с ним классов можно конструировать XML-документы, загружать и обращаться к данным, изменять данные и сохранять изменения.

Модель DOM для XML

Класс XML DOM является представлением XML-документа в памяти. Модель DOM позволяет читать, обрабатывать и изменять XML-документ программным образом. Класс **XmlReader** также читает XML; однако он обеспечивает некэшируемый однопроходный доступ только для чтения. Это значит, что у **XmlReader** нет возможности изменять значения атрибута, содержимое элемента, вставлять и удалять узлы. Изменение — основная функция модели DOM. Это стандартизованный, структурированный способ представления XML-данных в памяти, хотя на самом деле данные XML хранятся в файлах и пересылаются из других объектов в строковом виде. Далее приведен пример XML-данных.

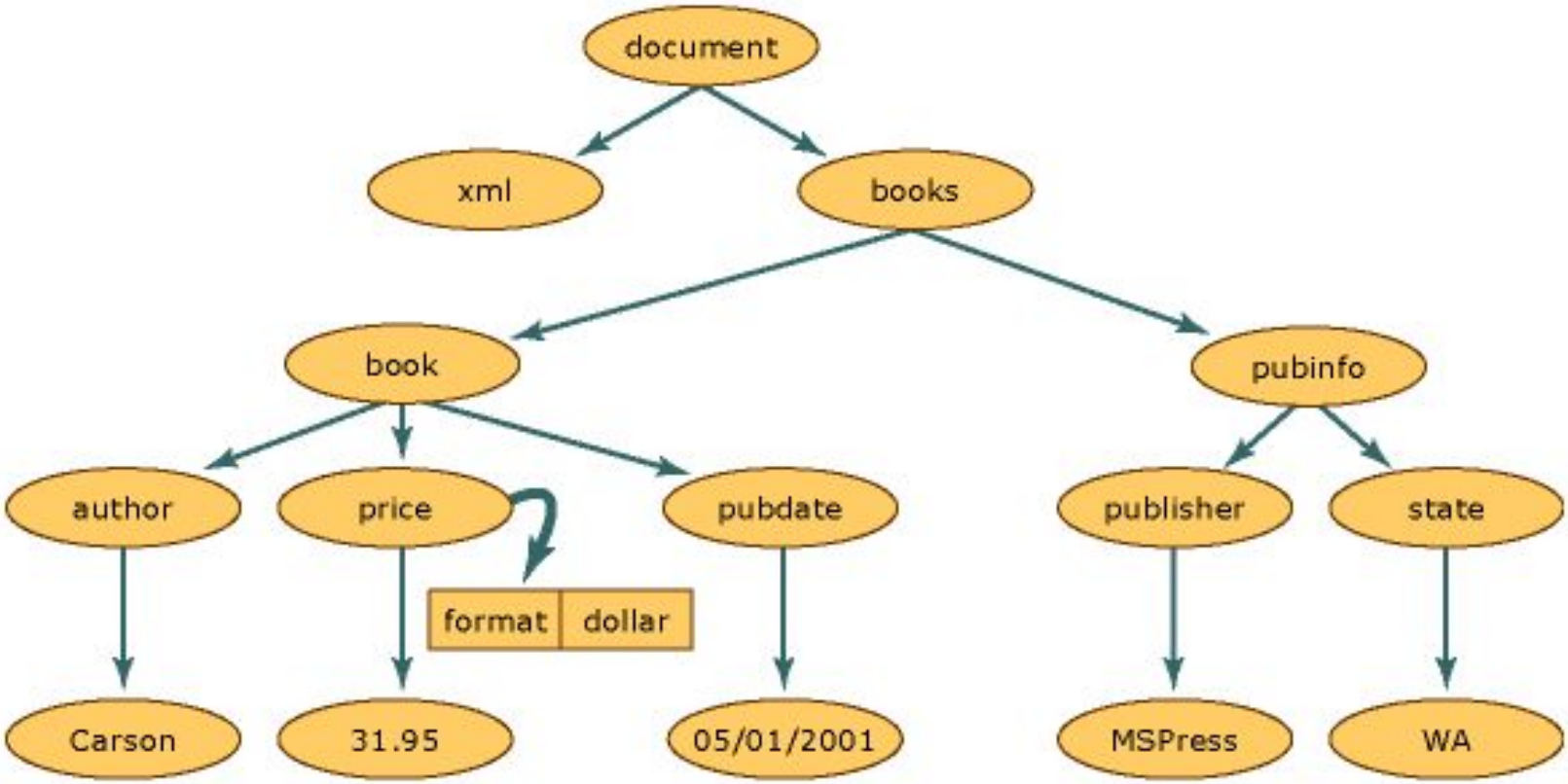
Модель DOM для XML

```
<?xml version="1.0"?>
<books>
 <book>
<author>Carson</author>
 <price format="dollar">31.95</price>
 <pubdate>05/01/2001</pubdate>
 </book>
<pubinfo>
<publisher>MSPress</publisher>
<state>WA</state> </pubinfo> </books>
```

Модель DOM для XML

Далее показано, какая структура будет создана в памяти, когда эти XML-данные считываются в модель структуры DOM.

Структура XML-документа



Модель DOM для XML

Каждый круг в данной иллюстрации представляет собой узел в структуре XML-документа, называемый объектом **XmlNode** .

Объект **XmlNode**является базовым объектом дерева DOM.

Класс **XmlDocument**, расширяющий класс **XmlNode**, поддерживает методы для выполнения операций над документом в целом (например, загрузки его в память или сохранения XML в файл).

Модель DOM для XML

Кроме того, **XmlDocument** предоставляет возможности для просмотра узлов всего XML-документа и выполнения операций над ними. И **XmlNode**, и **XmlDocument** обладают улучшенной производительностью, расширенной функциональностью и содержат методы и свойства, которые позволяют следующее.

- Получать доступ к DOM-специфичным узлам, например к узлам элементов, узлам ссылок на сущности и т. п., и изменять эти узлы.
- Получать целые узлы помимо содержащейся в них информации, например текста в узле элемента.

Модель DOM для XML

Объекты **Node** обладают набором методов и свойств, а также базовых, хорошо определенных характеристик.Вот некоторые из этих характеристик:

У каждого узла есть один родительский узел, то есть узел, находящийся непосредственно над данным.Единственный узел, не имеющий родителя — корневой узел документа, так как это узел верхнего уровня, содержащий сам документ и его фрагменты.

У большинства узлов может быть несколько дочерних узлов, то есть узлов, расположенных непосредственно под ними.Далее следует список типов узлов, которые могут иметь дочерние узлы:

Document

DocumentFragment

EntityReference

Element

Attribute

Модель DOM для XML

Узлы **XmlDeclaration**, **Notation**, **Entity**, **CDATASection**, **Text**, **Comment**, **ProcessingInstruction** и **DocumentType** не могут иметь дочерних узлов.

Узлы, находящиеся на одном уровне, — как узлы **book** and **pubinfo** на схеме, — называются одноуровневыми.

Модель DOM для XML

Одна из характеристик модели DOM — способ обработки атрибутов.Атрибуты не являются узлами, состоящими в родительских, дочерних и одноуровневых связях.Атрибуты считаются собственностью узла элемента и представляют собой пару «имя-значение».Например, если XML-данные представляют собой конструкцию format="dollar, связанную с элементом price, слово format является именем атрибута, а значением атрибутаformat является dollar.Для получения атрибута format="dollar" узла **price** можно воспользоваться методом **GetAttribute**, когда курсор расположен в узле элемента.

Модель DOM для XML

По мере считывания XML-документа в память создаются узлы.Узлы бывают разных типов.Правила и синтаксис XML-элемента отличаются от правил и синтаксиса инструкции по обработке. Поэтому по мере считывания разнообразных данных каждому узлу присваивается тип.Тип узла определяет его характеристики и функциональность.

Дополнительные сведения о типах узлов, создаваемых в памяти, см. в разделе Типы XML-узлов.

Модель DOM для XML

Модель DOM чрезвычайно полезна для считывания XML-данных в память, изменения их структуры, добавления и удаления узлов, изменения данных, принадлежащих узлу (например, текста, содержащегося в документе). Однако существуют и другие классы, которые в некоторых ситуациях работают быстрее модели DOM.

Классы **XmlReader** и **XmlWriter** предоставляют быстрый некэшируемый однопроходный потоковый доступ к XML. Если нужен произвольный доступ с моделью курсора и **XPath**, используется класс **XPathNavigator**.

Типы XML-узлов

Когда XML-документ считывается в память в виде дерева узлов, типы для узлов выбираются во время их создания. В модели XML DOM существует несколько типов узлов, определяемых консорциумом W3C. В следующей таблице перечислены типы узлов, объекты, назначаемые каждому типу узла, и дано краткое описание типов.

Типы XML-узлов

Тип узла модели DOM	Объект	Описание
Document	Класс XmlDocument	Контейнер для всех узлов в дереве. Он также называется корнем документа, что не всегда совпадает с корневым элементом.
DocumentFragment	Класс XmlDocumentFragment	Временный контейнер, содержащий один или несколько узлов, не имеющих древовидной структуры.
DocumentType	Класс XmlDocumentType	Представляет узел <!DOCTYPE…>.
EntityReference	Класс XmlEntityReference	Представляет текст нераскрытой ссылки на сущность.
Element	Класс XmlElement	Представляет узел элемента.
Attr	Класс XmlAttribute	Атрибут элемента.
ProcessingInstruction	Класс XmlProcessingInstruction	Узел инструкций по обработке.
Comment	Класс XmlComment	Узел комментария.

Типы XML-узлов

Тип узла модели DOM	Объект	Описание
Comment	Класс XmlComment	Узел комментария.
Text	Класс XmlText	Текст, принадлежащий элементу или атрибуту.
CDATASection	Класс XmlCDataSection	Представляет CDATA.
Entity	Класс XmlEntity	Представляет декларации <!ENTITY…> в XML-документе, полученные из встроенного DTD или из внешних DTD и сущностей параметров.
Notation	Класс XmlNotation	Представляет нотацию, объявленную в DTD.

Типы XML-узлов

В следующей таблице показаны дополнительные типы узлов, которые не определены консорциумом W3C, но доступны для использования в модели объектов Microsoft .NET Framework в виде перечислений **XmlNodeType**. Таким образом, для этих типов узлов отсутствует соответствующий столбец типа узла в модели DOM.

Типы XML-узлов

Тип узла	Описание
XmlDeclaration	Представляет узел декларации `<?xml version="1.0"…>`.
XmlSignificantWhitespace	Представляет значимые пробелы, то есть пробелы в смешанном содержимом.
XmlWhitespace	Представляет пробелы в содержимом элемента.
EndElement	Возвращается, когда модуль **XmlReader** достигает конца элемента. Пример XML-кода: **</item>** Дополнительные сведения см. в разделе Перечисление XmlNodeType.
EndEntity	Возвращается, когда модуль **XmlReader** достигает конца замещения сущности в результате вызова метода ResolveEntity. Дополнительные сведения см. в разделе Перечисление XmlNodeType.

Обработка XML

Понятие **обработка** означает **обработку информации**, что включает *ввод, проверку, организацию, хранение, поиск, преобразование и извлечение информации из данных*.

После того как вы создали модель данных в XML, важно подумать о различных приложениях, обрабатывающих данные, и определить, где именно в приложении необходима эта информация. Вы должны предоставить приложению возможность доступа к данным, а также различные способы подачи запросов и изменения данных.

Для обработки XML можно применить одну или несколько перечисленных ниже технологий: SAX, DOM, XPath и XSLT. У каждой технологии есть свои преимущества и недостатки, важно выбрать правильное решение для конкретной проблемы. В зависимости от вашего приложения для решения определенной бизнес-проблемы можно использовать совместно несколько технологий.

DOM как структура

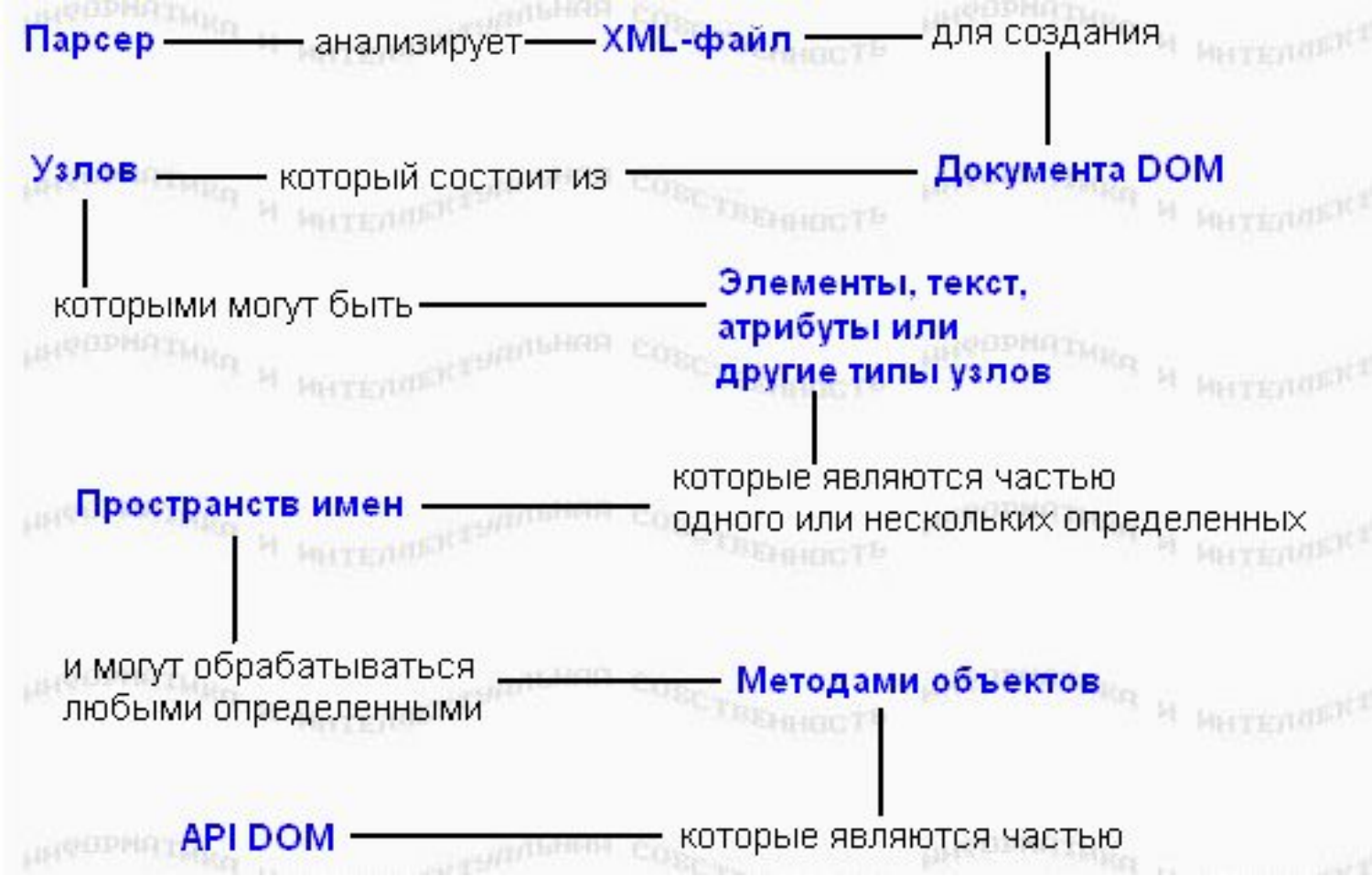
Прежде, чем начинать работу с DOM, стоит получить представление о том, что она на самом деле представляет. Объект DOM Document является коллекцией **узлов** или порций информации, организованных в иерархию. Эта иерархия позволяет разработчику двигаться по дереву в поисках нужной информации. Анализ структуры обычно требует, чтобы был загружен полный документ, и иерархия была построена до начала работы. Поскольку DOM основывается на иерархии информации, про нее говорят, что она **древовидно-базированная** или **объектно-базированная**.

DOM как структура

Для исключительно больших документов разбор и загрузка полного документа может быть медленной и ресурсоемкой, так что для работы с такими данными могут оказаться лучшими другие средства. **Событийно-базированные** модели, такие, как Simple API for XML (SAX), работают на потоке данных, обрабатывая его по мере поступления. Событийно-базированный API обходит необходимость построения дерева в памяти, но фактически не позволяет разработчику изменять данные в исходном документе.

С другой стороны, DOM также обеспечивает API, который позволяет разработчику добавлять или удалять узлы в любой точке дерева в надлежащим образом созданном приложении.

Карта DOM


Парсер
анализирует
XML-файл
для создания
Документа DOM
который состоит из
Узлов
которыми могут быть
Элементы, текст,
атрибуты или
другие типы узлов
которые являются частью
одного или нескольких определенных
Пространств имен
и могут обрабатываться
любыми определенными
Методами объектов
которые являются частью
API DOM

DOM как структура

Работа с DOM затрагивает несколько концепций, которые работают вместе. Вы изучите отношения между ними в этом учебнике.

Парсер - это программное приложение, которое предназначено для того, чтобы анализировать документ - в нашем случае XML-файл - и делать что-то определенное с его информацией. В событийно-базированном API, таком, как SAX, парсер события некоторому слушателю. В древовидно-базированном API , таком, как DOM, парсер строит в памяти дерево данных.

DOM как API

Начиная с DOM Уровня 1, DOM API содержит интерфейсы, которые представляют всевозможные типы информации, которые могут быть найдены в XML-документе, такие, как элементы и текст. Он также включает в себя методы и свойства, необходимые для работы с этими объектами.

Уровень 1 включает в себя поддержку XML 1.0 и HTML, в которой каждый элемент HTML представляется как интерфейс. Он включает в себя методы для добавления, редактирования, перемещения и чтения информации, содержащейся в узлах и т.д. Он, однако, не включает в себя поддержку пространств имен XML, которые обеспечивают возможность сегментировать информацию внутри документа.

DOM как API

Поддержка пространств была добавлена в DOM Уровня 2. Уровень 2 расширяет Уровень 1, позволяя разработчикам обнаруживать и использовать информацию пространств имен, которая может быть применима к узлу. Уровень 2 добавляет также несколько модулей поддержки Каскадируемых Таблиц Стилей, Cascading Style Sheets (CSS), событий и расширенных манипуляций с деревом.

DOM Уровня 3, в настоящее время находящийся на последнем этапе разработки, включает в себя улучшенную поддержку объекта Document (предыдущие версии оставляли это на усмотрение приложений, что делало затруднительным создание родовых приложений), расширенную поддержку пространств имен, и новые модули для загрузки и сохранения документов, проверки правильности и XPath, средства для выбора узлов, используемые в XSL Transformations и других технологиях XML.

Модуляризация DOM означает для разработчиков, что вы должны знать, поддерживаются ли те возможности, которые вы хотите использовать, той реализацией DOM, с которой вы работаете.

Базовый XML-файл

Примеры во всем этом учебнике используют XML-файл, который содержит приведенный ниже пример кода, представляющий заказы, проходящие через торговую систему. Кратко, основными частями XML-файла являются:

XML-объявление: Основное объявление <?xml version"1.0"?> определяет этот файл, как XML-документ. Не является общепринятым задание кодировки в объявлении, как показано ниже. Здесь не имеет значения, какой язык или кодировку использует XML-файл, парсер в состоянии читать его правильно, пока он понимает данную кодировку.

Базовый XML-файл

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE ORDERS SYSTEM "orders.dtd">
<orders> <order>
<customerid limit="1000">12341</customerid>
<status>pending</status>
<item instock="Y" itemid="SA15">
<name>Silver Show Saddle, 16 inch</name>
<price>825.00</price>
<qty>1</qty> </item>
<item instock="N" itemid="C49">
<name>Premium Cinch</name>
<price>49.00</price>
<qty>1</qty>
</item> </order> <order>
<customerid limit="150">251222</customerid>
<status>pending</status> <item instock="Y" itemid="WB78">
<name>Winter Blanket (78 inch)</name>
<price>20</price> <qty>10</qty> </item> </order> **</orders>**

Базовый XML-файл

Примеры во всем этом учебнике используют XML-файл, который содержит приведенный ниже пример кода, представляющий заказы, проходящие через торговую систему. Кратко, основными частями XML-файла являются:

XML-объявление: Основное объявление <?xml version"1.0"?> определяет этот файл, как XML-документ. Не является общепринятым задание кодировки в объявлении, как показано ниже. Здесь не имеет значения, какой язык или кодировку использует XML-файл, парсер в состоянии читать его правильно, пока он понимает данную кодировку.

Базовый XML-файл

В DOM работа с XML-информацией означает разбиение ее сначала по узлам.

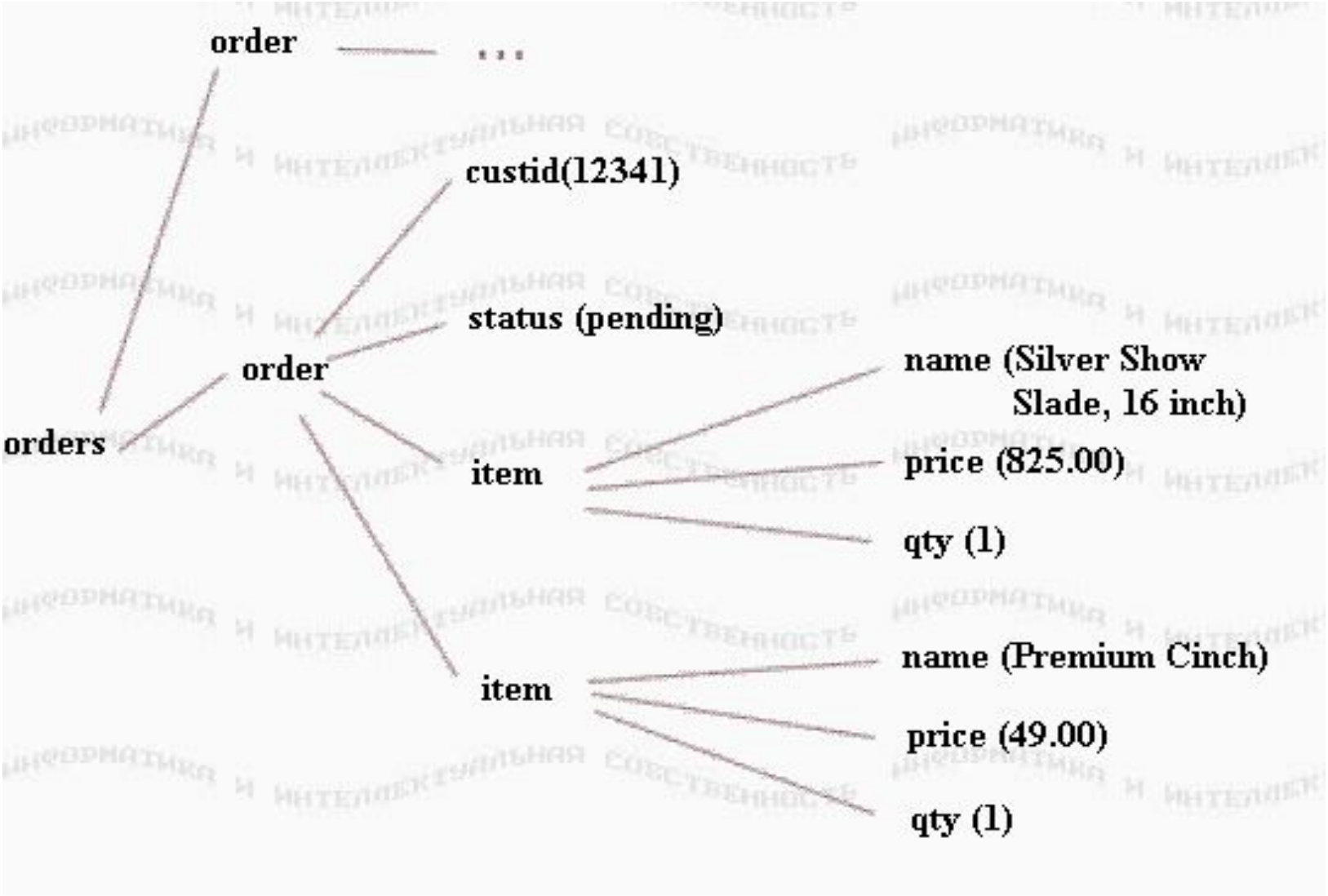
Различные типы узлов XML

Создание иерархии

DOM, в сущности, является коллекцией узлов. При том, что в документе потенциально содержатся разные типы информации, определено и несколько типов узлов.

При создании иерархии для XML-файла естественно выработать нечто концептуальное, наподобие приведенной выше структуры. Хотя она правильно описывает состав включенных в концепцию данные, она не дает описания данных с точки зрения их представления в DOM. Это потому, что она представляет *элементы*, а не *узлы*.

Различные типы узлов XML

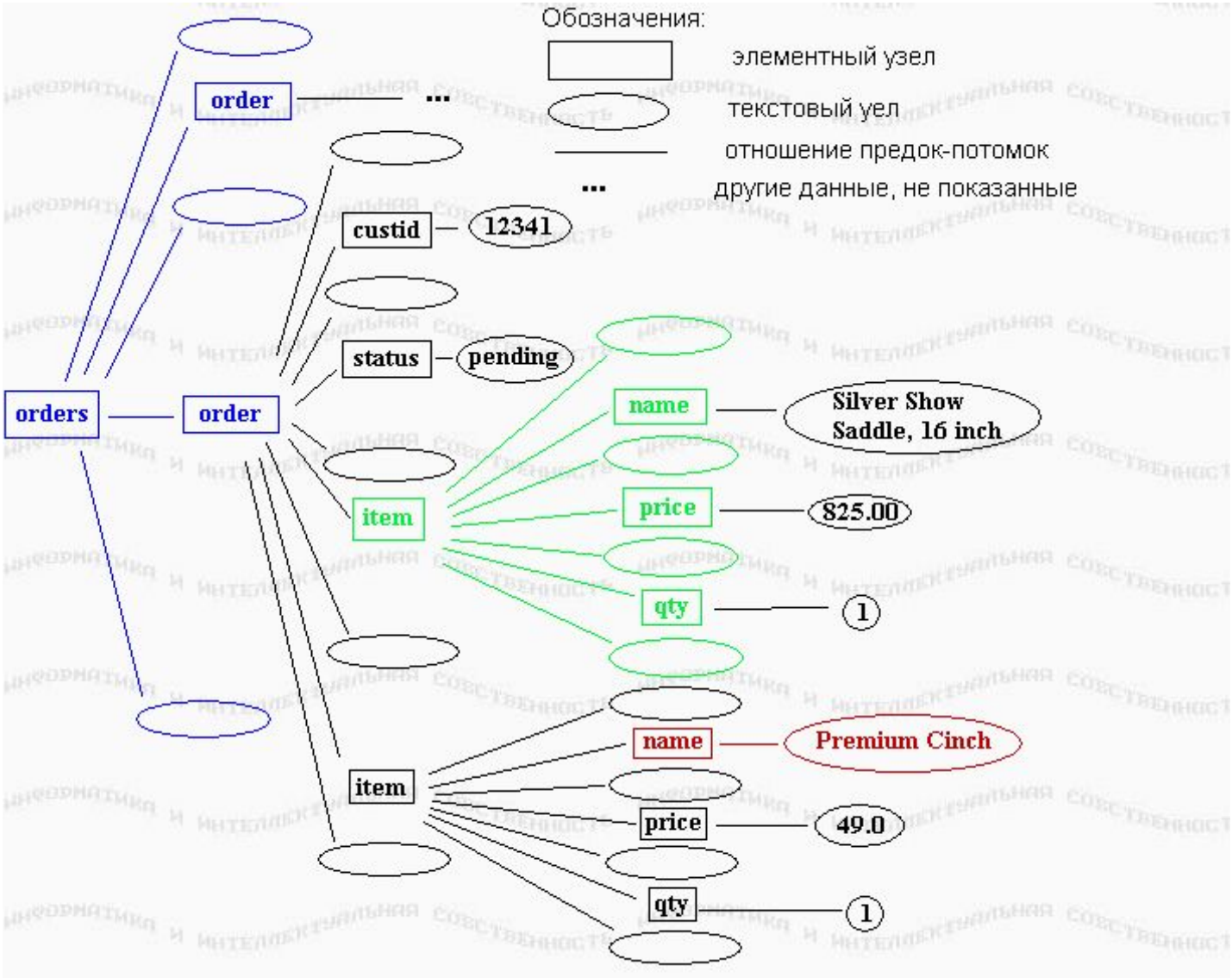


Различные типы узлов XML

Различие между элементами и узлами

Фактически, элементы являются только одним типом узлов, и они даже не выделены на предыдущем рисунке. Элементный узел - это контейнер для информации. Эта информация может быть другими элементными узлами, текстовыми узлами, узлами атрибутов или другого типа. Более правильная картина для документа показана ниже:

Различные типы узлов XML



Различные типы узлов XML

Прямоугольники представляют элементные узлы, а овалы представляют **текстовые узлы**. Если один узел содержит в себе другой, то последний рассматривается как **потомок** этого узла.

Заметьте, что элемент orders имеет не двух, а пять потомков: два элемента order и текстовые узлы между ними и вокруг них. Несмотря на то, что они не имеют содержимого, пропуски между элементами order составляют текстовый узел. Аналогично, item имеет семь потомков: name, price, qty и четыре текстовых узла вокруг них.

Заметьте также, что то, что может рассматриваться как содержимое элемента, например, "Premium Cinch", на самом деле является содержимым текстового узла, который является потомком элемента name.

Базовые типы узлов: документ, элемент, атрибут и текст

Наиболее распространенными типами узлов в XML являются:

Элементы: Элементы являются базовыми строительными блоками XML. Обычно элементы имеют потомков, которыми являются другие элементы, текстовые узлы или их комбинация. Элементные узлы также являются единственным типом узлов, имеющим атрибуты.

Атрибуты: Узлы атрибутов содержат информацию об элементном узле, но не рассматриваются как потомки элемента, как в:<customerid **limit="1000"** >12341</customerid>

Текст: Текстовый узел - это именно текст. Он может содержать информацию или только пропуск.

Менее распространенные типы узлов

Другие типы узлов используются не так часто, но все равно важны в некоторых ситуациях. В их число входят:

CDATA: Сокращение от Character Data (символьные данные), это узел, содержащий информацию, которая не должна анализироваться парсером. Вместо этого она должна просто передаваться как обычный текст. Например, в ней может быть записан HTML для специальных целей. При нормальных обстоятельствах процессор должен пытаться создать элементы из каждого записанного тега, который даже может не быть правильно форматированным. Эти проблемы могут быть обойдены применением секций CDATA. Эти секции записываются в специальной нотации:

<[CDATA[*Important: Please keep head and hands inside ride at <i>all times</i>. * ***]]>***

Менее распространенные типы узлов

Комментарии: Комментарии включают в себя информацию о данных и обычно игнорируются приложением. Они записываются как:**<!--** This is a comment. **-->**

Инструкции обработки: Инструкции обработки - это информация, специально адресованная приложению. Некоторыми примерами являются коды, которые должны быть выполнены, или информация о том, где найти таблицу стилей. Например:<? xml-stylesheet type="text/xsl" href="foo.xsl"? >

Менее распространенные типы узлов

Фрагменты документа: Чтобы быть правильно форматированным, документ должен иметь только один корневой элемент. Иногда при работе с XML должны временно создаваться группы элементов, для которых нет необходимости удовлетворять этому требованию. Фрагмент документа выглядит так:<item instock="Y" itemid="SA15"> <name>Silver Show Saddle, 16 inch</name> <price>825.00</price> <qty>1</qty> </item> <item instock="N" itemid="C49"> <name>Premium Cinch</name> <price>49.00</price> <qty>1</qty> </item>

Другие типы узлов включают в себя сущности, узлы ссылок на сущности и нотации. Одним из способов дополнительной организации XML-данных является применение пространств имен.

Пространства имен

Что такое пространство имен?

Одно из главных усовершенствований между DOM Уровня 1 и DOM Уровня 2 является добавление поддержки **пространств имен**. Поддержка пространств имен позволяет разработчикам использовать информацию из разных источников или в разных целях без конфликтов.

Концептуально пространства имен являются зонами, в которых все имена должны быть уникальны.

Например, я работаю в офисе и у меня то же имя, что и у клиента. Если я где-то в офисе, и секретарь говорит: "Ник, возьми трубку на 1-й линии", - каждый понимает, что она имеет в виду меня, потому что я нахожусь в "пространстве имен офиса". Аналогично, если она говорит "Ник звонит по 1-й линии", - каждый знает, что она говорит о клиенте, потому что звонящий находится вне

Пространства имен

С другой стороны, если я нахожусь вне офиса, и она делает такое же заявление, возможно недоразумение, поскольку существуют две возможности.

Те же проблемы возникают, когда XML-данные из разных источников комбинируются (как в информации о кредитоспособности в файле-примере, подробно рассматриваемом в этом учебнике позже).

Создание пространства имен

Поскольку идентификаторы для пространств имен должны быть уникальными, они обозначаются при помощи Унифицированных Идентификаторов Ресурсов, Uniform Resource Identifiers или URI. Например, пространство имен по умолчанию для данных примера будет обозначено при помощи атрибута xmlns:

<?xml version="1.0" encoding="UTF-8"?> <orders ***xmlns="http://www.nicholaschase.com/orderSystem.html"*** *> <order> <customerid limit="1000">12341<customerid> ... </orders>*

Создание пространства имен

Любые элементы, для которых не указано пространство имен, находятся в пространстве имен по умолчанию, http://www.nicholaschase.com/orderSystem.html. В действительности сам URI ничего не означает. Информация может находиться или не находиться по этому адресу, но что важно, так это то, что он уникален.

Важно отметить громадную разницу между пространством имен по умолчанию и отсутствием пространства имен вообще. В этом случает элементы, которые не имеют префикса пространства имен, находятся в пространстве имен по умолчанию. Если же не существует пространства имен по умолчанию, такие элементы находятся вне пространства имен.

Определение пространств имен

Для данных могут быть определены также и другие пространства имен. Например, созданием пространства имен rating вы можете добавить информацию оценки кредитоспособности в текст заказа, не беспокоясь об имеющихся данных.

Пространство имен вместе с алиасом создаются обычно (но не обязательно) в корневом элементе документа. Этот алиас используется как префикс для элементов и атрибутов - при необходимости, если используется более одного пространства имен, - чтобы задать правильное пространство имен.

Определение пространств имен

Рассмотрим код, приведенный ниже. Пространство имен и алиас, rating, использованы для создания элемента creditRating.

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

<orders xmlns="http://www.nicholaschase.com/orderSystem.html" **xmlns:rating="http://www.nicholaschase.com/rating.html"** > <order>

<customerid limit="1000"> 12341 <**rating:** creditRating>good</**rating:** creditRating> </customerid>

<status> pending </status>

<item instock="Y" itemid="SA15">

<name> Silver Show Saddle, 16 inch </name>

<price> 825.00 </price>

<qty> 1 </qty>

</item> ... </orders>

Информация пространства имен может быть получена для узла после того, как документ был разобран.

Разбор файла в документ

Трехшаговый процесс

Чтобы работать с информацией в XML-файле, файл должен быть разобран для создания **объекта Document**.

Объект Document является интерфейсом, так что его экземпляр не может быть создан непосредственно, обычно вместо этого приложение использует фабрику. Подробности этого процесса различаются от реализации к реализации, но идеи одни и те же. (Опять-таки, Уровень 3 стандартизирует эту задачу.) Например, в среде Java разбор файла является 3-шаговым процессом:

Создание DocumentBuilderFactory. Этот объект создает DocumentBuilder.

Создание DocumentBuilder. DocumentBuilder действитель но выполняет разбор для создания объекта Document.

Разбор файла для создания объекта Document.

Базовое приложение

Начнем с создания базового приложения, класса с именем OrderProcessor.

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import java.io.File;

import org.w3c.dom.Document;

public class OrderProcessor { public static void main (String args[]) { File docFile = new File("orders.xml"); Document doc = null;

*try { **DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance(); DocumentBuilder db = dbf.newDocumentBuilder(); doc = db.parse(docFile);** } catch (Exception e) { System.out.print("Problem parsing the file: "+e.getMessage()); } } }*

Разбор файла в документ

Сначала Java-код импортирует необходимые классы, а затем создает приложение OrderProcessor. Примеры в этом учебнике рассматривают один файл, так что для краткости приложение содержит прямую ссылку на него.

Так как объект Document может быть использован позже, приложение определяет его вне блока try-catch.

В блоке try-catch приложение создает объект DocumentBuilderFactory, который затем используется для создания DocumentBuilder. Наконец, DocumentBuilder разбирает файл для создания Document.

Установки парсера

Одно из преимуществ создания парсеров при помощи DocumentBuilder состоит в управлении различными установками парсера, создаваемого при помощи DocumentBuilderFactory. Например, парсер может быть установлен на проверку правильности документа:

```
... try { DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(true); DocumentBuilder db = dbf.newDocumentBuilder(); doc = db.parse(docFile); } catch (Exception e) { ...
```

Разбор файла в документ

Java-реализация DOM Уровня 2 обеспечивает управление параметрами парсера через следующие методы:

setCoalescing(): Определяет, превращает ли парсер узлы CDATA в текст и соединяет ли их с окружающими текстовыми узлами (если возможно). Значение по умолчанию - false.**setExpandEntityReferences():** пределяет, расширяются ли внешние ссылки на сущности. Если true, внешние данные вставляются в документ. Значение по умолчанию - true. (Приемы работы с внешними сущностями см. в *Ресурсы*.)

setIgnoringComments(): Определяет, игнорируются ли комментарии в файле. Значение по умолчанию - false.

setIgnoringElementContentWhitespace(): Определяет, игнорируются ли пропуски между элементами (аналогично тому, как браузер интерпретирует HTML). Значение по умолчанию -false.

setNamespaceAware(): Определяет, обращает ли парсер внимание на информацию пространства имен. Значение по умолчанию - false.

setValidating(): По умолчанию парсер не проверяет правильность документов. Установите здесь true для проверки правильности.

Редактирование документа

Просмотр содержимого XML-документ полезен, но когда вы имеете дело с полнофункциональным приложением, вам может понадобиться изменять данные, добавляя, перемещая или удаляя информацию. Возможность редактирования данных также важна при создании новых XML-документов. Простейшим из таких изменений является изменение текстового содержания элемента.

Нашей целью является изменить значение текстового узла элемента, в данном случае установкой status для каждого order в "processed", а затем вывести новые значения на экран.

Редактирование документа

Метод changeOrder() вызывается с передачей ему начального узла (root) в качестве параметра, а также имени изменяемого элемента и измененного значения.

changeOrder() сначала проверяет имя узла, чтобы увидеть, тот ли элемент, который редактируется. Если это так, приложению нужно изменить значение не этого узла, а его первого потомка, поскольку этот первый потомок является текстовым узлом, который на самом деле содержит содержимое элемента.

Редактирование документа

В противном случае приложение проверяет каждый потомок так же, как это делалось при прохождении по документу в первый раз.

Когда изменения выполнены, значение проверяются при помощи getElementsByTagName(). Этот метод возвращает список всех дочерних элементов с заданным именем, таким, как status. Приложение может затем проверить значения в списке, чтобы убедиться, что методchangeOrder() работает.

Добавление узлов: подготовка данных

Иногда необходимо не изменить существующий узел, а добавить узел, и у вас есть несколько способов сделать это. В нашем примере приложение вычисляет общую стоимость каждого заказа и добавляет в order элемент total. Оно получает общую стоимость, выбирая каждый заказ и проходя через все его составляющие, чтобы получить стоимость составляющей, а затем итоговую стоимость их всех. Затем приложение добавляет новый элемент в заказ.

Сначала приложение выбирает элементы order так же, как оно выбирало элементы status. Затем перебирает каждый из этих элементов.

Добавление узлов: подготовка данных

Для каждого из этих order приложению нужен NodeList из его составляющих item, так что приложение должно сначала преобразовать узел (Node) order в Element, чтобы использовать getElementsByTagName().

Приложение затем может перебрать составляющие item для выбранного order. Каждая из них преобразуется в Element, так что из него можно выбрать по имени price и qty. Приложение делает это при помощи метода getElementsByTagName(), и поскольку их всего по одному в каждом item, оно может прямо брать item(0), первую составляющую результирующего NodeList. Этот первый элемент представляет элемент price(или qty). Из него