Sun Microsystems

Web службы стали доступнее.

API и структуры языка Java™ для XML.
Техническая документация.

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U. S. A. 650-960-1300

2 издание, октябрь 2001.

Замечания по этой документации присылайте на xml-feedback@sun.com

Введение и краткий обзор.

Язык XML делает данные переносимыми. Платформа Java делает код переносимым. Интерфейс Java API для XML делает удобным использование самого XML. Объедините эти факторы вместе, и вы получите идеальное сочетание: переносимость данных, переносимость кода и простота использования. В действительности, с помощью API языка Java для XML вы можете использовать достоинства XML при помощи незначительного использования этого языка или совсем без него.

Предприятия быстро находят преимущества использования XML при интеграции данных как внутри самого предприятия — для совместного использования среди своих подразделений, так и вне него — для совместного использования с другими предприятиями. Благодаря интеграции данных, которую предлагает язык XML, он стал основой для вычислений, связанных с Web.

Крайне трудная часть развивающихся сетевых служб — планирование инфраструктуры, так называемой «системы трубопроводов», включающей безопасность, возможность передачи сообщений, распределенное управления транзакциями и управление пулом соединений. Еще одна сложность состоит в том, что сетевые службы должны обслуживать огромное количеством пользователей одновременно, поэтому приложения должны быть масштабируемыми. Этим требованиям в точности удовлетворяет платформа Java™ 2, Enterprise Edition (J2EE™). Добавьте ко всему этому то, что J2EE platform является проверенной технологией, включающей многочисленных производителей предлагающих совместимую продукцию в наши дни, и получится очевидный факт, состоящий в том, что платформа J2EE является наилучшим инструментальным комплексом разработки сетевых служб. И с помощью новых API языка Java для XML разработка сетевых служб становится все легче и легче.

Цель данной статьи – прояснить, как действуют прикладные программные интерфейсы (API) Java для XML, и как они облегчают написание сетевых приложений. Эта статья описывает каждый API в отдельности, и затем представляет сценарий, показывающий их работу вместе. Также упоминаются и другие технологии, доступные в наше время, и их возможное использование в сочетании с прикладными программными интерфейсами Java для XML. В конце статьи вы найдете глоссарий, который поможет вам разобраться во всех акронимах и разъяснит терминологию.

Подробная информация о различных API языка Java для XML доступна по адресу:

http://java.sun.com/xml

Что такое XML?

XML(eXtensible Markup Language – расширяемый язык разметки) – это стандартный, независимый от системы способ представления данных. Как и HTML (HyperText Markup Language – язык разметки гипертекста), XML заключает данные внутрь тэгов, однако между этими языками существуют значительные различия. Во-первых, тэги XML связаны с содержанием текста, заключенного между ними, в то время как тэги HTML определяют способ отображения текста. Следующий пример отображает прайс-лист с названием и ценой двух сортов кофе:

```
<priceList>
  <coffee>
    <name>Mocha Java</name>
    <price>11.95</price>
  </coffee>
  <coffee>
    <name>Sumatra</name>
    <price>12.50</price>
  </coffee>
</priceList>
```

Тэги <coffee> and </coffee> указывают синтаксическому анализатору, что информация между ними касается кофе. Два других тэга внутри тэгов <coffee> определяют, что заключенная в них информация – это название кофе и его цена за 1 фунт. Благодаря тому, что XML тэги показывают содержимое и структуру данных, заключенных в них, они делают возможными такие операции, как архивирование и поиск.

Второе значительное отличие между HTML и XML состоит в том, что тэги XML являются расширяемыми, позволяя вам писать собственные тэги для описания вашего содержания. С HTML вы ограничены тем набором тэгов, которые были предопределены спецификацией HTML.

Благодаря расширяемости, которую предоставляет XML, вы можете создавать тэги, которые вам требуются для данного конкретного типа документа. Для определения тэгов используется язык схем XML.

Схема описывает структуру набора XML документов и может быть использована для создания оглавлений XML документов. Вероятно, самым широко используемым языком схем является Document Type Definition (язык классов в документах). Схема, написанная на этом языке, называется DTD. Следующий пример DTD задает тэги, которые используются в прайс-листе из предыдущего примера. Он определяет четыре тэга (элемента) и далее описывает тэги, которые могут потребоваться (или обязательно потребуются) в других тэгах. DTD также задает иерархию XML документа, включая порядок, в котором тэги должны использоваться.

```
<!ELEMENT priceList (coffee)+>
<!ELEMENT coffee (name, price) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT price (#PCDATA) >
```

Первая строка в этом фрагменте задает элемент самого высокого уровня, priceList, который означает, что все остальные тэги в документе будут расположены между тэгами <priceList> и </ priceList>. Первая строка также сообщает, что элемент priceList обязан включать в себя один или более элементов coffee (на это указывает знак плюс). Вторая строка определяет, что каждый элемент coffee, в свою очередь, должен содержать два элемента: name и price. Третья и четвертая строки указывают, что данные между тэгами <name> и </name>, а также <price> и </price> представляют собой символьные данные, которые должны быть проанализированы. Собственно текстом прайс-листа являются название и цена каждого из сортов кофе.

Что делает XML переносимым?

Именно описания DTD, подобные DTD прайс-листа, обеспечивают переносимость XML формата данных. Если приложение получает priceList в XML формате, и имеет в своем распоряжении priceList DTD, то оно может обработать данные согласно правилам, определенным в DTD. Например, имея описания DTD priceList, анализатор будет знать структуру и вид содержимого для любого XML документа, основанного на данном DTD. Если анализатор призван проверить документ на соответствие, то он поймет, что документ не подходит, если в нем присутствуют элементы, не включенные в DTD, такие как <tea>, к примеру, или скажем, если элемент price стоит раньше, чем name.

Популярность XML, как метода обмена данными, определяют также и другие особенности этого языка. С одной стороны, он использует текстовый формат, который может быть прочитан как человеком, так и текстовым редактором. Программы могут анализировать и обрабатывать XML документы, и человек может проверить их в случае возникновения ошибки. С другой стороны, благодаря тому, что XML не содержит команд форматирования текста, он может отображаться различными способами. Разделение данных и команд форматирования позволяет публиковать одну и ту же информацию в различных источниках.

Краткий обзор прикладных программных интерфейсов (API) языка Java для XML.

Прикладные программные интерфейсы Java для XML позволяют вам писать Web-приложения целиком и полностью на языке программирования Java. Эти интерфейсы делятся на две обширные категории: одна из них работает с XML документами напрямую, другая же – с помощью подпрограмм.

Прикладные программные интерфейсы (API), ориентированные на прямую работу с документами:

- Java ™ API for XML Processing (JAXP) – обрабатывает XML документы, используя различные синтаксические анализаторы
- Java ™ Architecture for XML Binding (JAXB) – преобразовывает элементы XML в классы языка программирования Java

Прикладные программные интерфейсы (API), ориентированные на работу с документами при помощи подпрограмм:

- Java ™ API for XML Messaging (JAXM) – позволяет посылать SOAP (Simple Object Access Protocol) сообщения в Internet стандартным способом
- Java ™ API for XML Registries (JAXR) – обеспечивает стандартный способ получения доступа к производственному учету и совместно используемой информации
- Java ™ API for XML-based RPC (JAX-RPC) – посылает запросы процедур SOAP удаленным сторонам в Internet и получает результаты.

Возможно, самое главное свойство всех API языка Java for XML состоит в том, что они поддерживают промышленные стандарты, таким образом, обеспечивая возможность для взаимодействия. Различные группы

по стандартам сетевых взаимодействий, такие как WWW-консорциум (World Wide Web Consortium, W3C) и Организация по развитию стандартов структурированной информации (Organization for the Advancement of Structured Information Standards, OASIS), определили способы, как надо действовать, чтобы компании, следующие эти стандартам, могли заставить данные и приложения работать вместе.

Еще одна характерная особенность различных API языка Java для XML – то, что они обеспечивают большую гибкость, и пользователи используют эту гибкость в работе. Например, код JAXP может применять различные средства для обработки XML документов, а код JAXM может оперировать различными протоколами передачи поверх SOAP. Конструкторы также отличаются гибкостью. Прикладные программные интерфейсы Java для XML определяют строгие требования совместимости, для того чтобы не только гарантировать соответствие реализаций стандартным функциональным возможностям, но также дать разработчикам большой простор для исполнения реализаций, приспособленные под конкретные задачи.

Следующие главы рассматривают каждое из этих API и дают общее представление об их использовании.

JAXP

Краткий обзор.

JAXP(The Java™ API for XML Processing) упрощает обработку XML данных с помощью программам, написанными на языке программирования JAVA. JAXP усиливает стандартные анализаторы SAX (Simple API for XML Parsing) и DOM (Document Object Model) таким образом, что вам предоставляется выбор между анализом данных как потока событий или созданием объектной структуры для них. JAXP версии 1.1 также поддерживает стандарт XSLT(XML Stylesheet Language Transformations), предоставляя управление представлением данных и давая возможность преобразования информации в другие XML документы или другие форматы, такие как HTML. JAXP также обеспечивает поддержку пространства имен (namespace), позволяя вам работать с DTD. В противном случае, это могло бы привести к конфликту имен.

Обладая гибкостью, JAXP дает возможность использовать в вашей программе любой XML-совместимый синтаксический анализатор. Это возможно благодаря так называемому слою подключений, который разрешает вам подключать прикладные программные интерфейсы SAX или DOM. Слой подключений также позволяет подключать XSL процессор, что в свою очередь позволяет вам управлять отображением XML данных. JAXP 1.1 Reference Implementation (доступно по адресу http://java.sun.com/xml) предоставляет XSLT процессор Xanon и синтаксический анализатор Crimson, разработанные совместно Sun и организацией Apache Software Foundation, которая предоставляет программное обеспечение с открытым исходным кодом.

Прикладной программный интерфейс (API) SAX

SAX создает прикладной программный интерфейс для синтаксического анализатора, основанного на событиях. Основанный на событиях анализатор просматривает XML документ от начала и до конца, посылая сообщение запущенному приложению каждый раз, когда он встречает синтаксическую конструкцию. Анализатор SAX использует методы ContentHandler интерфейса для отправки сообщений программе. Например, когда анализатору попадается символ (“<”), он вызывает метод StartElement; когда ему встречаются символьные данные, он вызывает метод characters; когда он доходит до символа следующего за слешем (“</”), он вызывает метод EndElement и так далее. Для иллюстрации вышесказанного, рассмотрим фрагмент XML документа из первой части и проследим построковую работу синтаксического анализатора. (Для простоты, опустим вызовы метода ignorableWhiteSpace)

```
<priceList>       [ анализатор вызывает startElement ]
    <coffee>        [ анализатор вызывает startElement ]
        <name>Mocha Java</name>      [ анализатор вызывает startElement, characters, and endElement ]
        <price>11.95</price>     [ анализатор вызывает startElement, characters, and endElement ]
    </coffee>     [ анализатор вызывает endElement ]
```

Методы, вызываемые анализатором по умолчанию, ничего не делают, таким образом, вам нужно написать подкласс соответствующих методов, чтобы достигнуть нужных вам результатов. К примеру, вы хотите получить цену одного фунта за Mocha Java. Вам следует написать класс, расширяющий DefaultHandler (заданное по умолчанию значение ContentHandler) в котором вы опишите ваши собственные методы startElement и characters.

Сначала вам нужно создать объект SAXParser из объекта SAXParserFactory. Вам потребуется вызвать метод parse, анализируя прайс-лист и исключение для вашего нового заглавного класса (с его новыми методами startElement и characters). В этом примере прайс-лист представляет собой файл, однако метод parse может также проверять и другие источники входных данных, включая InputStream, URL и InputSource.

```
SAXParserFactory factory = SAXParserFactory.newInstance( ) ;
SAXParser saxParser = factory.newSAXParser( ) ;
saxParser.parse( "priceList.xml" , handler) ;
```

Результат работы метода parse зависит, разумеется, от того, как были применены handler методы. Анализатор SAX будет считывать файл priceList.xml сточка за строчкой, применяя соответствующие методы. Кроме уже упомянутых, анализатор будет использовать и другие методы, такие как startDocument , endDocument , ignorableWhiteSpace , и processingInstructions, но эти методы сохраняют свое начальное значение, и, следовательно, не станут ничего делать.

Следующий пример определения методов иллюстрирует один из путей применения characters и startElement, для поиска цены Mocha Java. Благодаря способу функционирования синтаксического анализатора SAX, эти два

метода одновременно ищут элемент name, строку “Mocha Java”, и элемент price непосредственно следующий за Mocha Java. Эти методы используют три флага, чтобы следить за встречаемыми условиями. Помните, что анализатор SAX вызовет оба метода более одного раза, прежде чем встретит нужную цену.

```
public void startElement( . . . , String elementName, . . . ) {
  if(elementName.equals("name" ) ) {
    inName = true;
  } else if(elementName.equals("price" ) && inMochaJava ){
      inPrice = true;
      inName = false;
  }
}

public void characters(char [] buf, int offset, int len) {
  String s = new String(buf, offset, len) ;
  if (inName && s.equals("Mocha Java" ) ) {
     inMochaJava = true;
     inName = false;
  } else if ( inPrice) {
       System.out.println("The price of Mocha Java is: " + s) ;
       inMochaJava = false;
       inPrice = false;
       }
  }
}
```

Как только анализатор найдет элемент coffee Mocha Java , будут вызваны следующие методы:

```
next invocation of startElement --inName is true
next invocation of characters --inMochaJava is true
next invocation of startElement --inPrice is true
next invocation of characters --prints price
```

Синтаксический анализатор SAX во время работы с данными XML может производить проверку на соответствие документа DTD. SAX проверит, создан ли он при помощи конструктора объектов SAXParserFactory, что и активизирует проверку документа. Для SAXParserFactory это делается следующим кодом:

```
Factory.setValidating(true) ;
```

Таким образом, для того чтобы анализатор понимал, какой DTD использовать для проверки, XML документ должен передать DOCTYPE. Объявление DOCTYPE должно быть подобно этому:

```
<!DOCTYPE PriceList SYSTEM "priceList. DTD">
```

Прикладной программный интерфейс к объектной модели документа (DOM API)

DOM API (Document Object Model – объектная модель документа), описанная W3C DOM Working Group, представляет собой набор интерфейсов для построения объектного представления в форме дерева анализируемого XML документа. Построив DOM один раз, вы можете управлять ею с помощью DOM методов, таких как insert и remove (вставка и удаление), абсолютно аналогично тому, как вы бы работали с любыми другими деревьями. Таким образом, в отличие от синтаксического анализатора SAX, анализатор DOM позволяет произвольно доступаться к каким-либо конкретным частям документа. Другое отличие состоит в том, что с помощью SAX вы можете только считывать XML документы, а DOM дает вам возможность построить в памяти объектную модель и работать уже с ней, добавляя новые элементы и удаляя ненужные.

В предыдущем примере мы использовали анализатор SAX для поиска фрагмента во всем документе. Использование анализатора DOM потребовало бы построения объектной модели всего документа в памяти, что, в общем-то, менее эффективно для поиска всего лишь нескольких небольших фрагментов, особенно при работе с большими документами. В следующем примере с помощью DOM мы добавим в прайс несколько новых марок кофе. Использование для этой цели анализатора SAX невозможно по причине того, что он может лишь считывать данные.

Предположим, что в прайс-лист требуется добавить кофе Kona. Для этого нужно прочитать прайс-лист в XML формате с помощью DOM и добавить в него новый элемент типа coffee с его названием и ценой. Следующий код создает объект DocumentBuilderFactory, который в дальнейшем используется для создания объекта DocumentBuilder *builder.*Затем код вызывает анализатор параметра *builder*, передавая файл priceList.xml.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance( ) ;
DocumentBuilder builder = factory.newDocumentBuilder( ) ;
Document document = builder.parse( "priceList. xml" ) ;
```

На этом этапе мы получаем в памяти DOM-представление документа. Следующий код добавляет в прайс-лист новый кофе (с названием “Kona” и ценой 13.50). Так как новый кофе требуется добавить именно перед элементом “Mocha Java”, то сначала нужно получить список элементов name и перебирать названия, пока не встретится “Mocha Java”. Используя интерфейс Node(узел), включенный в пакет org.w3c.dom, код создает для нового элемента coffee объект Node, а также новые объекты Node для названия и цены. Элементы name и price состоят из символьных данных, поэтому для каждого из них код создает TextNode объекты и добавляет текстовые узлы в узловое представление элементов name и price.

```
NodeList list = document.getElementsByTagName("name" ) ;
Node thisNode = list.item(0) ;
/ /  loop through list
Node thisChild = thisNode.getChildNode() ;
if(thisNode.getFirstChild() instanceof org.w3c.dom.TextNode) {
   String data = thisNode.getFirstChild().getData( ) ;
}
if (data. equals( " Mocha Java" ) ) { / / новый узел будет вставлен перед Mocha Java
   Node newNode = document.createElement("coffee" ) ;
   Node nameNode = document.createElement("name" ) ;
   TextNode textNode = document.createTextNode("Kona" ) ;
   nameNode.appendChild(textNode) ;

  Node priceNode = document.createElement("price" ) ;
  TextNode tpNode =document.createTextNode("13.50" ) ;
  priceNode. appendChild( tpNode) ;

  newNode. appendChild( nameNode) ;
  newNode. appendChild( priceNode) ;
  thisNode. InsertBefore( newNode, thisNode) ;
}
```

Проверку синтаксический анализатор DOM выполняет аналогично анализатору SAX: перед началом использования DOM вызывается setValidating(true) в конструкторе DOM и создается ваш синтаксический анализатор, после чего вы можете быть уверенны, что XML документ, который вы анализируете, соответствует DTD, декларированному в DOCTYPE.

Пространство имен XML.

Все имена в DTD являются уникальными, избегая, таким образом, двусмысленности. Однако если отдельный XML документ связан более чем с одним DTD, то есть вероятность, что два или более DTD содержат одно и то же имя. Поэтому нужно выделить пространство имен для каждого DTD, чтобы анализатор знал, какое из определений использовать при обработке запросов от конкретного DTD.

Существует стандартная система обозначений для объявления пространства имен XML, которое, как правило, имеет место в корневом элементе XML документа. В следующем примере для объявления пространства имен XML указание xmlns определяет nsName как пространство имен, и nsName ставится в соответствие URL с реальным пространством имен:

< priceList xmlns: nsName="myDTD.dtd"
xmlns: otherNsName="myOtherDTD.dtd" >
. . .
< / priceList>

Внутри документа принадлежность элемента к соответствующему пространству имен определяется следующим образом:

<nsName:price> . . .

Чтобы научить SAX или DOM распознавать пространства имен, нужно использовать метод setNamespaceAware(true) вашего ParserFactory. После этого любой анализатор, созданный этим конструктором синтаксических анализаторов, будет знать соответствующие пространства имен.

Прикладной интерфейс программирования к языку описания стилей XML **(XSLT API).**

XSLT (XSL Transoformation), описанная W3C XSL Working Group, описывает язык преобразования XML документов в другие XML документы или другие форматы. Чтобы выполнить преобразование, как правило, нужно подключить таблицу стилей, написанную на XSL(XML Stylesheet Language – язык таблиц стилей). Таблицы стилей XSL определяют способ отображения XML данных. Чтобы выполнить преобразование, XSLT использует команды форматирования в таблице стилей. Преобразованный документ может представлять собой как XML документ, так и документ в ином формате, например, в формате HTML.

JAXP поддерживает XSLT с помощью пакета javax.xml.transform, который позволяет подключать XSLT. Подпакеты содержат SAX-, DOM- и потоковые API, которые дают возможность преобразовывать DOM-деревья в события SAX. Следующие два примера иллюстрируют, как из DOM-дерева можно создать XML документ, и после чего преобразовать полученный XML документ в HTML, используя таблицу стилей XSL.

Преобразование DOM-дерева в XML документ.

Чтобы преобразовать DOM-дерево из предыдущей главы в XML документ, приведенный ниже, код первым делом создает Transformer object , который будет выполнять преобразование.

```
TransformerFactory transFactory = TransformerFactory.newInstance();
Transformer transformer = transFactory.newTransformer();
```

Используя корневой узел DOM-дерева, Следующая строка создает DOMSource объект как источник преобразования.

```
DOMSource source = new DOMSource(document);
```

А эта часть кода конструирует объект StreamResult для получения результатов преобразования и трансформации дерева в XML.

```
File newXML = new File("newXML.xml");
FileOutputStream os = new FileOutputStream(newXML) ;
StreamResult result = new StreamResult(os) ;
transformer.transform(source, result) ;
```

Преобразование XML документа в HTML документ.

XSLT можно также использовать для того, чтобы трансформировать только что полученный XML документ newXML.xml в HTML с помощью таблицы стилей. При создании таблицы стилей для связи с XSL конструкциями используется пространство имен XML. Каждая таблица, к примеру, содержит корневой элемент, определяющий язык стилей, что иллюстрирует следующая строка.

<xsl:stylesheet version= "1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

Когда вы ссылаетесь на какую-либо отдельную конструкцию языка стилей, вы используете префикс пространства имен, сопровождаемый двоеточием и конструкцией, которую вы хотите использовать. Например, приведенная ниже часть таблицы, показывает, что имена должны быть заключены внутри HTML-таблицы.

```
<xsl:template match= "name">
    <tr> <td>
      <xsl:apply-templates/>
    </td> </tr>
</xsl:template>
```

Следующий пример определяет, что XML данные преобразованы в HTML, и таким образом все сорта кофе размещены в таблице.

```
< xsl:stylesheet version= "1.0" xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match= "priceList">
    <html> <head>Coffee Prices</head>
      <body>
        <table>
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match= "name">
    <tr><td>
      <xsl:apply-templates/>
    </td></tr>
  </xsl:template>
  <xsl:template match= "price">
    <tr><td>
      <xsl:apply-templates/>
    </td></tr>
  </xsl:template>
</xsl:stylesheet>
```

Чтобы выполнить преобразование, нужно получить XSLT конвертор и применить с его помощью таблицы стилей к XML данным. Данный код получает конвертор, для чего сначала инициализирет TransformerFactory объект, затем считывает таблицу стилей и XML файлы, создает файл для записи HTML кода, и в итоге получает Transformer объект *transformer* из объекта *tFactory* типа TransformerFactory .

```
TransformerFactory tFactory = TransformerFactory.newInstance() ;
String stylesheet = "prices.xsl";
String sourceId = "newXML.xml";
File pricesHTML = new File("pricesHTML.html");
FileOutputStream os = new FileOutputStream(pricesHTM);
Transformer transformer = tFactory.newTransformer(new StreamSource(stylesheet) ) ;
```

Преобразование заканчивается вызовом метода transform, который помещает преобразованные данные в выходной поток (output stream).

```
transformer.transform(new StreamSource(sourceId), new StreamResult(os));
```

JAXB

JAXB предлагает быстрый и удобный способ создания двухстороннего преобразования между XML документами и классами языка Java. Задавая DTD, компилятор JAXB создает набор классов Java, содержащий весь код, который необходим для анализа XML документов, основанных на данной схеме. Разработчик, использующий созданные классы, может строить дерево объектов Java, изображающих XML документ, обрабатывать содержимое этого дерева и заново создавать XML документы из него.

Для того чтобы начать использование приложения JAXB, все, что вам нужно, это схема, и для данной версии JAXB это должна быть DTD. Вы можете написать свою схему DTD, или же вы можете получить ее где-либо еще, например, из стандартного репозитория DTD через JAXB.

Как только у вас появляется схема DTD, вы связываете ее с набором классов следующим образом:

1. Запишите связующую схему, которая содержит инструкции для привязки схемы к классам. Связующая схема написана на языке связи, основанном на XML и содержащемся в JAXB.
2. Запустите компилятор схем, который берет DTD и связывающую схему и создает из них классы. Каждый класс, создаваемый компилятором схем, обладает средствами для считывания и изменения параметров. После того, как экземпляр класса создан и наполнен информацией, вы можете использовать эти средства доступа для обращения к данным. Набор средств доступа называется property (свойством).

Создание классов из DTD.

В качестве примера генерирования классов из DTD рассмотрим следующую DTD, которая называется priceList.dtd.

```
<!ELEMENT priceList (coffee)+ >
<!ELEMENT coffee (name, price) >
<!ELEMENT name (#PCDATA) >
<!ELEMENT price (#PCDATA) >
```

JAXB компилятор схем достаточно мощный, для того проведения правильного анализа DTD и связующей схемы, которая устанавливает только основной элемент документа. Все, что вам нужно определить в связывающей схеме, – то, что элемент price преобразовывается к состоянию, которое принимает и возвращает значение типа BigDecimal (большого целого):

```
...
<element name="priceList" type="class" class="PriceList" root="true" />
<element name="price" type=" value" convert="BigDecimal" />
<conversion name="BigDecimal" type="java.math.BigDecimal" />
    ...
```

Из данной DTD и связующей схемы генератор схем создает класс PriceList и класс Coffee.

Класс PriceList содержит конструктор и параметр List, с которым связан элемент "кофе".

Класс Coffee включает в себя конструктор и переменную для представления названия кофе и параметра для представления цены.

Средствами доступа к элементу "цены" являются функции:

```
BigDecimal getPrice();
void setPrice(BigDecimal x);
```

Как класс PriceList, так и класс Coffee содержит средства для демаршализации, подтверждения и размещения. Демаршализация – процесс построения объектного представления данных XML. Подтверждение – это процесс проверки соответствия объектов техническим требованиям DTD. Размещение – это процесс создания данных XML из объектного представления.

Создание объектных представлений данных XML

После построения классов, вы можете писать Java приложение, используя классы, и формировать объектные представления XML документов, которые являются действующими относительно DTD. Каждый объект соответствует элементу в XML документе. Подобным образом, каждый объект является экземпляром класса из набора созданных классов. Так как эти объекты отображают как документ, так и классы, то у вас есть два

способа построения дерева объектов Java: демаршализацией доступного XML документа или обработкой объектов классов. Таким образом, JAXM дает возможность вам как обрабатывать существующие XML документы, так и создавать новые данные XML с помощью обработки созданных классов.

Предположим, что у вас есть следующий XML документ:

```
<priceList>
    <coffee>
        <name>Arabica</name>
        <price>13.50</price>
    </coffee>
    <coffee>
        <name>Mocha Java</name>
        <price>11.95</price>
    </coffee>
    <coffee>
        <name>Sumatra</name>
        <price>12.50</price>
    </coffee>
 </priceList>
```

Для того чтобы демаршализовать этот XML документ, вы создаете из него поток ввода и вызываете метод демаршализации класса PriceList:

```
FileInputStream fis =new FileInputStream("priceList.xml");
PriceList myPrices =PriceList.unmarshal(fis);
```

Теперь у вас есть дерево объектов Java, в корне которого находится объект myPrices.

Допустим, вы хотите создать собственный список цен на кофе как XML документ. Сначала вы порождаете дерево объектов с помощью обработки данных, а затем размещаете это дерево в XML документе. Для того чтобы получить дерево объектов при помощи реализации, создайте новый объект типа PriceList, затем получите из него список объектов Coffee, создайте новый объект типа Coffee и в заключение добавьте его в список:

```
PriceList myNewPrices = new PriceList();
List listOfCoffees = myNewPrices.getCoffees();
Coffee zCoffee = new Coffee();
zCoffee.setName("Zapoteca");
zCoffee.setPrice("15.00");
listOfCoffees.add(zCoffee);
```

Представив данные XML в форме объектного дерева, вы можете работать с ними так же, как если бы вы работали с объектами Java. Таким образом, JAXB предоставляет интерфейс программирования Java для данных XML и также позволяет цельную интеграцию данных XML в приложения Java.

Доступ к данным из дерева объектов

Предположим, что вы хотите поменять цену кофе Mocha Java в первом созданном вами объекте. Все, что вам необходимо сделать, – это найти элемент Mocha Java в списке кофе и установить новую цену с помощью вызова процедуры setPrice для объекта Coffee:

```
List coffees =myPrices.getCoffees();
for (ListIterator i =coffees.listIterator();i.hasNext();){
Coffee myCoffee =(Coffee)i.next();
    if (myCoffee.getName().equals("Mocha Java")){
        myCoffee.setPrice("12.50");
    }
}
```

Формирование XML документов из дерева объектов

Использовали ли вы демаршализацию или реализацию для построения объектного представления данных, вы можете разместить объекты в XML документе. Это означает, что JAXB тоже позволяет вам создавать новые качественные XML документы, являющиеся действительными относительно схемы.

Для того чтобы разместить ваше измененное дерево объектов в XML документе, создайте XML файл и выходной поток для него и вызовите разделяющую функцию для объекта myNewPrices:

```
File newPrices =new File("newPriceList.xml");
FileOutputStream fos = new FileOutputStream(newPrices);
myNewPrices.marshal(fos);
```

Резюме

По существу, JAXB обеспечивает взаимодействие между технологией Java и XML. Так же, как XML документ - пример схемы, объект Java - пример класса. Таким образом, JAXB позволяет вам создавать объекты Java на таком же абстрактном уровне, как и данные XML. Представление информации таким способом дает вам возможность управлять ей таким же образом, как если бы вы манипулировали объектами Java, что облегчает создание приложений для обработки данных XML. Как только вы представите ваши данные в форме объектов Java, к ней легко можно будет обратиться. К тому же, после работы с данными, вы можете записать объекты Java в новый полноценный XML документ. JAXB представляет собой самый простой способ доступа к данным XML, при котором вы освобождаетесь от написания кода для анализа и обработки данных, для того чтобы писать приложения, которые будут действительно использовать данные.

Различия между JAXP и JAXB

JAXP и JAXB служат весьма различным целям. Выбор архитектуры API зависит от требований вашего приложения. Одним из достоинств JAXP является то, что он позволяет вам анализировать и обрабатывать данные из того же самого набора прикладных программных интерфейсов (API). Если вы хотите выхватить всего лишь часть данных из большого документа, вам следует использовать SAX парсер по причине того, что он очень быстро анализирует данные как поток. Если ваш документ не слишком большой, и вы собираетесь добавить или удалить информацию из него, то вам следует использовать модель DOM. Хотя дерево DOM может потребовать большого объема памяти, API модели DOM включает в себя обыкновенные функции управления деревом. Если же вы намереваетесь изменить формат данных, то вам следует использовать JAXP, который содержит трансформирующий API и XSLT в эталонным исполнении, позволяющие вам преобразовывать XML документы, события SAX или деревья DOM. JAXP дает вам возможность выбора, проверять достоверность данных или нет.

Если вы хотите построить объектное представление XML данных, а также обойти ограничения памяти DOM, то вам также следует использовать JAXB. Классы, созданные с помощью JAXB, не обладают возможностью управления деревьями, а это приводит к тому, что дерево объектов JAXB занимает небольшой объем памяти. Другим достоинством объектного дерева такого типа является возможность соединять деревья из условия, чтобы дочерний объект мог иметь более одного родительского. Кроме того, обработка данных с помощью JAXB осуществляется также быстро, как и обработка анализатором SAX, т.к. созданные классы содержат всю логику DTD, таким образом, избегая динамической интерпретации, которую анализатор SAX должен выполнить. То обстоятельство, что JAXB нуждается в DTD, делает его менее гибким, но это требование обеспечивает обработку только правильных данных. Эта гарантия очень важна, особенно в случае получения приложением информации из другого источника. Если документ не содержит DTD, он не может определить содержание данных и метод их обработки. Другое преимущество JAXB перед JAXP состоит в том, что он позволяет определять, каким образом ваш код будет генерироваться из DTD, включая типы данных, которые связующий элемент будет принимать и возвращать.

Используйте JAXB в случае, когда вы хотите:

- Обращаться к данным в памяти, но вам не нужна возможность обработки деревьев
- Обрабатывать только верные данные
- Преобразовывать данные к различным типам
- Создавать объектные представления данных XML

Используйте JAXP в случае, когда вы хотите:

- Иметь гибкость в отношении способа доступа к данным: либо последовательно с помощью SAX, либо произвольно в памяти с помощью DOM
- Использовать ваш код обработки для документов, построенных на различных DTD
- Анализировать документы, которые не обязательно являются правильными
- Применять XSLT преобразования
- Вставлять или удалять элементы из дерева объектов, представляющего XML данные

JAXM

Краткий обзор.

API языка Java для обмена XML сообщениями (The Java ™ API for XML Messaging, JAXM) предоставляет стандартный способ отправки XML документов через Internet. Он основан на протоколе SOAP версии 1.1 и SOAP в спецификации Attachments и может быть расширен для работы с протоколами сообщений более высоких уровней, такие как ebXML.

Как правило, предприятия пользуются службой своего поставщика услуг по обмену сообщениями. Поставщик услуг выполняет закулисную работу, необходимую для транспортировки и направления сообщений. При использовании провайдера по обмену сообщениями все JAXM сообщения идут через него, поэтому, когда фирма посылает сообщение, то оно сначала направляется к провайдеру отправителя, затем к провайдеру получателя и, наконец, попадает к намеченному адресату. Также возможно направить сообщение к вспомогательным получателям, прежде чем оно дойдет до последнего адресата.

Провайдер обмена сообщениями, проводящий сообщения, может уделить внимание служебным деталям, таким как назначение идентификатора сообщений, хранение сообщений и отслеживание того, было ли сообщение доставлено до этого. Провайдер обмена сообщениями может также заново послать сообщение, которое не дошло до места назначения при первой попытке доставки. Вся прелесть такой службы передачи сообщений состоит в том, что клиент, использующий технологию JAXM ("JAXM client"), находится в полном неведении относительно того, что провайдер делает в фоновом режиме. Этот клиент JAXM просто вызывает процедуры JAXM, и провайдер в соответствии со структурой сообщения выполняет все действия.

Обычно предприятие пользуется услугами провайдера для передачи сообщений, но также возможно передавать JAXM сообщения и без него. В этом случае, клиент JAXM (называемый автономным) ограничен посылкой сообщений типа "точка-точка" непосредственно в сетевую службу, выполняющей функции передачи сообщений запросов и ответов. Передача ответов и запросов происходит синхронно, то есть, запрос послан, и его ответ получен в одной при той же операции. Сообщение запрос-ответ передается в объект SOAPConnection, путем вызова процедуры SOAPConnection.call, которая посылает сообщение и останавливает свою работу до тех пор, пока не получит ответ. Автономный клиент может управлять только клиентские задачи, то есть отправлять запросы и получать на них ответы. Напротив, клиент JAXM, пользующийся услугами провайдера для обмена сообщений, может выполнять как клиентские, так и серверные (сервисные) задачи. Исполняя роль клиента, он может посылать запросы; в роли сервера, он может получать запросы, обрабатывать их и посылать ответы на них.

Обычно передача сообщений JAXM обычно происходит внутри контейнера, как правило, являющегося сервлетом (servlet) или контейнером J2EE™. Сетевая служба, которая использует провайдера передачи сообщений и применяется в контейнере, обладает возможностью отправки односторонних сообщений: это означает, что вы можете получать запросы и отсылать через некоторое время на них ответы, являющимися также односторонними сообщениями.

JAXM сообщение состоит из двух частей, обязательной SOAP части и дополнительной, прикрепленной части. SOAP часть, состоящая из объекта SOAPEnvelope содержит элементы SOAPHeader и SOAPBody. Объект SOAPBody может хранить XML документы в теле посылаемого сообщения. Если вы хотите послать не XML документ, то ваше сообщение должно будет хранить еще и прикрепленную часть, наряду с SOAP частью. На содержание прикрепляемой части ограничений не существуют, поэтому вы можете посылать изображения или любой другой документ, включая и XML.

Установление соединения

Первое, что надо сделать JAXM клиенту, это установить связь, будь то либо соединение SOAPConnection, либо ProviderConnection.

Установление связи "точка-точка"

Автономный клиент ограничен использованием объекта SOAPConnection, который является соединением типа "точка-точка", идущим напрямую от отправителя к получателю. Все подключения JAXM создаются конструктором соединений. В случае объекта SOAPConnection, конструктор является объектом SOAPConnectionFactory. Клиент получает заданное по умолчанию действие для объекта SOAPConnectionFactory, вызывая следующую строку программы:

```
SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();
```

Клиент может использовать объект factory для создания объекта SOAPConnection:

```
SOAPConnection con =factory.createConnection();
```

Установка соединения с провайдером сообщений

Для того чтобы пользоваться услугами провайдера сообщений, приложение должно получить ProviderConnection объект, который предпочтительнее должен быть соединением с провайдером сообщений, а не с определенным адресатом. Существует два способа получения ProviderConnection объекта, один из них схож со способом получения автономным клиентом SOAPConnection объекта. Этот способ включает в себя получение экземпляра подразумеваемой реализации для объекта ProviderConnectionFactory, который затем используется для создания соединения:

```
ProviderConnectionFactory pcFactory  = ProviderConnectionFactory.newInstance();
ProviderConnection pcCon = pcFactory.createConnection();
```

Переменная pcCon представляет собой соединение с заданной по умолчанию реализацией провайдеру JAXM сообщений.

Второй способ создать объект ProviderConnection – получить объект ProviderConnectionFactory, введенный для создания соединения с определенным провайдером сообщений. Следующий код демонстрирует получение такого объекта ProviderConnectionFactory и его использование для установки связи. Первые 2 строки используют JNDI API для поиска необходимого объекта ProviderConnectionFactory в службы имен, где он был внесен в список с именем "CoffeeBreakProvider". Передавая это логическое имя в качестве аргумента, поиск возвращает объект ProviderConnectionFactory, с которым было связано логическое имя. Возвращаемое значение является объектом языка Java, который должен быть ограничен до объекта ProviderConnectionFactory для того, чтобы он мог использоваться для установки соединения. Третья строка использует JAXM метод для фактического создания подключения:

```
Context ctx = getInitialContext();
ProviderConnectionFactory pcFactory = (ProviderConnectionFactory) ctx.lookup("CoffeeBreakProvider");
ProviderConnection con = pcFactory.createConnection();
```

Элемент con типа ProviderConnection символизирует соединение к провайдеру компании Coffee Break.

Создание сообщения

Как и соединения, сообщения создаются конструктором. И аналогично случаю с конструкторами соединений, объекты MessageFactory так же могут быть получены двумя способами. Первый способ состоит в получении экземпляра заданной по умолчанию реализации класса MessageFactory. Этот экземпляр может затем использоваться для создания основного SOAPMessage объекта.

```
MessageFactory messageFactory  = MessageFactory.newInstance();
SOAPMessage m = messageFactory.createMessage();
```

Все SOAPMessage объекты, которые создает messageFactory, включая элемент m в предыдущей строке кода, будут основными сообщениями протокола SOAP. Это значит, что у них не будет заранее определенных заголовков.

Одним из факторов гибкости JAXM API является возможность особого использования SOAP заголовка. Например, ebXML или BizTalk протоколы могут быть основаны на передаче сообщений по протоколу SOAP. Такое использование SOAP данной группой стандартов или индустрией, называется конфигурацией или профилем (profile). При втором способе создания объекта типа MessageFactory, вы используете метод createMessageFactory объекта ProviderConnection и задаете для него профиль. SOAPMessage объекты, созданные получившимся MessageFactory объектом, будут поддерживать особенную конфигурацию. Например, в следующем фрагменте кода, в котором идентификатор schemaURI является URI схемы с требуемой конфигурацией, объект m2 будет поддерживать профиль сообщений, который подается в createMessageFactory.

```
MessageFactory messageFactory2 = con.createMessageFactory(<schemaURI>);
SOAPMessage m2 = messageFactory2.createMessage();
```

Оба новых SOAPMessage объекта – m и m2 – автоматически содержат нужные элементы SOAPPart , SOAPEnvelope, и SOAPBody, а также дополнительные элемент SOAPHeader (который включен для удобства). Элементы SOAPHeader и SOAPBody изначально пустые, и в следующих разделах будут описаны стандартные способы для добавления содержимого.

Заполнение сообщения

Информация может быть занесена в объект SOAPPart, в один или более объект AttachmentPart или же в обе части сообщения.

Заполнение части SOAP сообщения

Как было раннее установлено, все сообщения содержат объект SOAPPart, который включает в себя элемент SOAPBody и элемент SOAPEnvelope, содержащий параметр SOAPHeader. Один из способов заполнения SOAP части сообщения, – это создание объекта SOAPHeaderElement или объекта SOAPBodyElement и добавление XML документа, который вы получаете с помощью функции SOAPElement.addTextNode. Первые три строки следующего фрагмента кода обращаются к телу объекта SOAPBody, который используется для того, чтобы создать новый элемент SOAPBodyElement и добавить его к телу. Аргумент, передаваемый функции createName, является Name объектом, определяющим добавление элемента SOAPBodyElement. Последняя строка передает XML строку функции addTextNode:

```
SOAPPart sp = m.getSOAPPart();
SOAPEnvelope envelope = sp.getSOAPEnvelope();
SOAPBody body = envelope.getSOAPBody();
SOAPBodyElement bodyElement = body.addBodyElement(envelope.createName("text",
                                        "hotitems","http://hotitems.com/products/gizmo");
bodyElement.addTextNode("some-xml-text");
```

Другой способ состоит в заполнении элемента SOAPPart с помощью передачи элемента javax.xml.transform.Source, который может быть SAXSource, либо DOMSource , либо StreamSource объектом. Исходный объект содержит информацию для SOAP части сообщения, а также информацию, необходимую для того, чтобы вести себя как исходный вход. Объект StreamSource будет содержать информацию в форме XML документа; объекты SAXSource, либо DOMSource, будут включать в себя информацию и команды для преобразования ее в XML документ.

Следующие фрагменты кода иллюстрируют заполнение объекта DOMSource. Первый шаг – получить объект SOAPPart из элемента SOAPMessage. Затем код использует средства JAXP для создания XML документа, который надо добавить. Он использует элемент DocumentBulderFactory для получения объекта DocumentBulder. Затем он анализирует заданный файл для формирования документа, который будет использоваться для инициализации объекта DOMSource. В заключение, код передает элемент DOMSource объекта domSource функции SOAPPart.setContent:

```
SOAPPart soapPart = message.getSOAPPart();

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("file:///foo.bar/soap.xml");
DOMSource domSource = new DOMSource(doc);

soapPart.setContent(domSource);
```

Заполнение прикрепляемой части сообщения

Элемент Message может и не содержать присоединенных частей, но если в нем есть что-либо не являющееся XML документом, то он должен включать прикрепляемую часть. Этот элемент может иметь в своем составе любое количество прикрепленных частей, и они могут содержать любую информацию, начиная с обычного текста, и кончая изображениями. В следующем фрагменте кода содержимое является рисунком формата JPEG, чей URL используется для инициализации объекта dh, являющегося переменной типа javax.activation.DataHandler. Переменная m типа Message создает объект attachPart типа AttachmentPart, который инициализируется обработчиком данных, имеющем на входе URL картинки. В заключении, сообщение присоединяет объект attachPart к себе.

```
URL url = new URL("http://foo.bar/img.jpg");
DataHandler dh = new DataHandler(url);
AttachmentPart attachPart = m.createAttachmentPart(dh);
m.addAttachmentPart(attachPart);
```

Объект SOAPMessage может также задавать содержание объекта AttachmentPart с помощью передачи элемента Object и типа его содержания функции createAttachmentPart.

```
AttachmentPart attachPart = m.createAttachmentPart("content-string", "text/plain");
m.addAttachmentPart(attachPart);
dsf
```

Третий вариант подразумевает создание пустого AttachmentPart объекта и последующей передачи элемента Object и его типа функции AttachmentPart.setContent. В этом фрагменте кода элемент Object является переменной типа ByteArrayInputStream, инициализированной изображением jpeg.

```
AttachmentPart ap = m.createAttachmentPart();
byte[] jpegData = ...;
ap.setContent(new ByteArrayInputStream(jpegData), "image/jpeg");
m.addAttachmentPart(ap);
```

Отправка сообщения

После того как вы заполнили объект SOAPMessage, вы готовы его послать. Автономный клиент использует вызов SOAPConnection процедуры для отправки сообщения. Это функция посылает сообщение и затем блокируется до тех пор, пока не получит ответ. Параметрами вызова этой функции являются отправляемое сообщение и объект Endpoint, содержащий URL получателя.

```
SOAPMessage response = soapConnection.call(message, urlEndpoint);
```

Приложение, оперирующее с провайдером сообщений, использует функцию ProviderConnection для отправки сообщения. Эта процедура посылает сообщение асинхронно, что означает, что она отправляет сообщение и сразу же возвращается к работе. Ответ, если таковой существует, будет отправлен в течение отдельной операции позже. Заметим, что эта функция получает лишь один параметр – отправляемое сообщение. Провайдер сообщений использует заголовочную информацию для определения адресата.

```
providerConnection.send(message);
```

JAXR

Краткий обзор.

Java™ API для XML Registries (JAXP) предоставляет удобный путь для доступа к стандартным деловым регистрационным формам через Интернет. Деловые регистрационные формы часто описываются как "желтые страницы" электронной коммерции, потому что они содержат списки предприятий и продукции или услуг, предлагаемых этими предприятиями. JAXR позволяет разработчикам писать Java приложения единым образом, используя деловые регистрационные формы, основанные на открытых стандартах (таких как ebXML), или промышленных синдикативных описаний (таких как UDDI).

Предприятия могут регистрироваться в регистрационных формах или находить другие компании, с которыми им хотелось бы сотрудничать. К тому же, они могут сделать публичными свои материалы или найти материалы, опубликованные другими организациями. Группы норм содержат DTD для конкретных типов XML документов, и две компании могут, к примеру, принять DTD для своих стандартных промышленных форм заказа. Благодаря тому, что DTD поддерживается стандартной деловой регистрационной формой, обе организации могут использовать JAXR для доступа к ней.

Регистрации становятся все более важной частью сетевых служб, потому что они позволяют предприятиям динамично сотрудничать в свободной взаимовыгодной форме. Соответственно потребность в JAXR, который предоставляет предприятиям доступ к стандартным регистрационным формам с помощью Java, также возрастает.

Использование JAXR.

В следующих главах приводятся примеры двух распространенных способов использования регистрационных форм. Это сделано для того, чтобы показать, как можно использовать возможности JAXR более полно.

Регистрация организации.

Организация, использующая Java платформу для электронных сделок, может использовать JAXR для регистрации в стандартной регистрационной форме. Она должна ввести свое имя, описание и несколько определяющих принципов для облегчения поиска. Это проиллюстрировано на примере, который сначала создает RegistryServise объект *rs* и затем использует его для создания BusinessLifeCycleManager объекта *lcm*. Сеть кафе The Coffee Break, обозначена с помощью Organization объекта *org*, которому компания The Coffee Break присваивает свое имя, описание и свое положение в NAICS (the North American Industry Classification System – промышленная классификационная система Северной Америки). После этого, элемент org, содержащий свойства и классификацию для The Coffee Break, добавляется в объект Collection *orgs*. В конечном счете, orgs сохраняется с помощью lcm, который будет управлять объектами Organization, заключенными в orgs.

```
RegistryService rs = connection.getRegistryService();

BusinessLifeCycleManager lcm = rs.getBusinessLifeCycleManager();
Organization org = lcm.createOrganization("The Coffee Break");
org.setDescription("Purveyor of only the finest coffees. Established 1895");

ClassificationScheme cScheme =
  lcm.createClassificationScheme("ntis-gov:naics",
    "North American Industry Classification System");

javax.xml.registry.infomodel.Key cKey =

  lcm.createKey("uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2");
cScheme.setKey(cKey);
```

```
Classification classification = (Classification)
  lcm.createClassification(cScheme,
    "Snack and Nonalcoholic Beverage Bars","722213");

Collection classifications = new ArrayList();
classifications.add(classification);

org.addClassifications(classifications);
Collection orgs = new ArrayList();
orgs.add(org);
lcm.saveOrganizations(orgs);
```

Поиск регистрационных форм.

Компания может использовать JAXR для поиска других компаний. Ниже приведенный пример использует BusinessQueryManager объект *bqm* для поиска компании The Coffee Break. Перед тем как bqm сможет вызвать метод findOrganizations, программа должна определить критерии поиска. В данном случае, для поиска используются три из шести возможных параметров, так как третий, пятый и шестой параметры имеют значение null, а потому не накладывают ограничений на поиск. Первый, второй и четвертый параметры являются Collection объектами, с определенными параметрами *findQualifiers* и *namePatterns*. Единственный элемент в *findQualifiers* – это String, указывающий, что не следует выводить организацию, если ее название не удовлетворяет с учетом регистра одному из названий, указанных в namePatterns. Этот параметр, также являющийся Collection объектом с одним элементом, определяет, что мы ищем компанию, название которой содержит слово “Coffee”. Другой Collection объект – это *classifications,* который был определен при регистрации The Coffee Break. Регистрация классификаций была проиллюстрирована в предыдущем примере, в котором была произведена регистрация The Coffee Break.

```
BusinessQueryManager bqm = rs.getBusinessQueryManager();

// Define find qualifiers
Collection findQualifiers = new ArrayList();
findQualifiers.add(FindQualifier.CASE_ SENSITIVE_ MATCH);
Collection namePatterns = new ArrayList();
namePatterns.add("%Coffee%"); //Find orgs with name containing Coffee

//Find using only the name and the classifications
BulkResponse response = bqm.findOrganizations(findQualifiers, namePatterns, null,
                                                classifications,null,null);
Collection orgs = response.getCollection();
```

JAXR также поддерживает запросы SQL при поиске регистрации. Как показывает следующий фрагмент, это делается с помощью объекта DeclarativeQueryManager.

```
DeclarativeQueryManager dqm = rs.getDeclarativeQueryManager();
Query query = dqm.createQuery(Query.QUERY_TYPE_SQL,
  "SELECT id FROM RegistryEntry WHERE name LIKE %Coffee%" +
  "AND majorVersion >= 1 AND" +
  "(majorVersion >= 2 OR minorVersion >= 3)");
BulkResponse response2 = dqm.executeQuery(query);
```

Объект *response2* типа BulkResponse содержит значение id (uuid) для каждого вхождения в RegistyEntry, содержащего слово “Coffee” в своем названии.

Чтобы обеспечить внутреннюю связь между JAXP клиентом и регистрационной программой, отсылка сообщений производится с помощью JASM. Это полностью делается внутри сценариев так, что пользователь JAXR не видит этого механизма.

JAX-RPC

Краткий обзор.

Java™ API для XML-based RPC (JAX-RPC) дает возможность писать Java приложения, использующие XML для выполнения удаленного вызова процедур (RPC – Remote Procedure Call).

Язык программирования Java уже имеет два других API для создания процедур удаленного вызова: Java IDL и Remote Method Invocation (RMI). Все три возможности имеют API для размещения и демаршализации аргументов, а также для передачи их процедурам. Различие состоит в том, что JAX-RPC основан на XML, что является более удобным для сетевых служб. Java IDL основан на COBRA (Common Object Request Broker Architecture) и использует OMG IDL (Object Management Group's Interface Definition Language). RMI основан на RPC, при котором и вызывающий и вызываемый методы должны быть написаны на языке Java, впрочем, с RMI через IIOP вызываемая процедура может быть написана на ином языке. Sun планирует продолжать поддержку технологий COBRA и RMI и развивать технологию JAX-RPC, поскольку каждый из этих сервисов имеет свое назначение и свой круг пользователей.

Все разновидности RPC достаточно сложны, и включают в себя отображение и обратное отображение типов данных, а так же размещение и демаршализацию аргументов. Однако это происходит внутри сценариев и незаметно для пользователя. JAX-RPC продолжает эту модель, и клиенту, использующему XML-based RPC из языка программирования Java, не нужно работать с XML или осуществлять какое-либо отображение напрямую.

Использование JAX-RPC.

JAX-RPC упрощает использование сетевых служб, а также их разработку, особенно при использовании J2EE платформы. Основанные на RCP сетевые службы, по существу, являются набором процедур, которые могут быть вызваны удаленным клиентом через Интернет. В свою очередь служба представляет собой серверное приложение, расположенное на сервере среди доступных клиенту процедур. Например, распространенный RPC Web-сервис stock quote, который через SOAP получает запрос на цену наличного товара и возвращает ее, также используя SOAP.

Сетевая служба должна быть доступна потенциальным клиентам, что может быть достигнуто, к примеру, с помощью WSDL (Web Services Description Language). После этого пользователь (Web-клиент) может осуществить поиск WSDL документа и получить доступ к сервису. Пользователь, использующий Java, задействует JAX-RPC, для того чтобы послать запрос службе, основанной на Java платформе, которая может быть или не быть определенной. Связь возможна и в том случае, если клиент, использующий другую платформу, запрашивает службу, основанную на платформе Java.

Хотя JAX-RPC использует SOAP для вызова удаленных процедур, пользователь JAX-RPC не видит этих технических деталей. Таким образом, изнутри, JAX-RPC, на самом деле представляет собой специализированную форму SOAP протокола, в отличие от JAXM, который является более устойчивой формой протокола SOAP, предоставляя разработчику все возможности. Это особенно заметно, когда протокол более высокого уровня, такой как ebXML, наложен на SOAP.

Вот список возможностей JAXM, которые JAX-RPC, вообще говоря, не поддерживает:

- Асинхронные сообщения
- Маршрутизация сообщений для нескольких адресатов
- Отправка сообщений с гарантированной доставкой.

JAX-RPC предпочтителен для тех случаев, когда требуется избежать сложности работы с SOAP, и когда связь через RPC модель является достаточной. Важно заметить, что независимо от того, что используется, JASM или JAX-RPC, с помощью Java можно удобно работать с сообщениями в формате XML.

Пример сценария

Краткий обзор.

Следующий сценарий демонстрирует использование Java API для XML, а так же их совместную работу. Достоинство Java APIs для XML состоит в том, что во многих случаях они предоставляют альтернативные пути, позволяя, таким образом, приспособить приложение для каких-либо конкретных нужд. Эта часть содержит несколько примеров, в которых можно несколькими способами достигать одних и тех же целей, обращая внимание на достоинства и недостатки каждого из них.

Сценарий.

Предположим, что хозяин сети кафе The Coffee Break желает расширить число продаваемых им сортов кофе. Он дает задание своему менеджеру найти новых поставщиков и получить их прайс-листы. Компания сможет изучить цены и принять решение о сотрудничестве с некоторыми из них. Менеджер передает распоряжение инженеру по программному обеспечению, который в свою очередь решает использовать для этой цели поиск с помощью регистрационной формы ebXML, в которой The Coffee Break уже зарегистрирована.

Инженер пользуется JAXR, чтобы запросить оптовых поставщиков кофе. JAXR отсылает сообщения при помощи JAXM, который гарантирует отправление и распознавание запроса.

Регистрационная книга ebXML получает запрос и запускает JAXR-код с полученными параметрами для анализа зарегистрированных организаций. По завершению поиска регистрационная книга возвращает список оптовых поставщиков кофе.

Следующий шаг инженера – запросить с помощью JAXM у каждого из поставщиков прайс-лист с ценами на кофе. Она пишет программу, которая соединяется со службами приема-отправки сообщений, таким образом, получая возможность сделать запрос. Затем она создает JAXM-сообщение, добавляет запрос и отправляет его.

Каждый поставщик получает запрос, и перед тем как отправить текущий прайс, он вызывает с помощью JAX-RPC сервис stock quotes, чтобы получить последние цены на имеющийся в наличии товар, предназначенный для крупных сделок. Получив ответ, он посылает самые последние ценовые обновления TheCoffee Break в XML формате. Оптовые фирмы используют формат XML, потому что он удобен для них и их клиенты могут работать с ним, даже если они используют различные информационные системы.

Сравнение цен и заказ кофе.

Инженер решает использовать JAXB для анализа прайс-листов. Список поставщиков, возвращенный регистрационной книгой, содержит информацию о DTD поставщиках, и что удобно, все они используют стандартную форму прайс-листа. Благодаря единому DTD оптовых компаний инженер может построить набор классов из этого DTD. (Иначе, она могла бы использовать для этой цели SAX или DOM.) Ее программа будет рассматривать каждый сорт кофе как объект, обладающий параметрами name и price. После инициализации классов, программа берет цены из объекта Coffee и сравнивает цены разных компаний.

Когда владелец и менеджер The Coffee Break, выберут поставщика, основываясь на работе, проделанной инженером, они будут готовы послать заказ. При помощи JAXB инженер создает новую XML форму заказа, основанную на классах, построенных из DTD прайс-листа. Эта форма, содержащая лишь те сорта кофе, которые владелец желает приобрести, будет отправлена поставщикам через JAXM. Каждая компания вышлет уведомление о получении заказа также через JAXM.

Продажа кофе через Интернет.

Между тем, The Coffee Break готовится расширить количество сортов продаваемого кофе. Это потребует переделать их прайс-лист в HTML формат для сайта компании. Но перед этим надо будет определить, что опубликовывать. Инженер использует все те же объекты, которые она создала для сравнения цен и составления формы оптового заказа, для того чтобы взять каждую из цен и повысить ее на 25%, дабы получить цену, по которой The Coffee Break будет продавать свою продукцию. После некоторых преобразований прайс-лист превратится в он-лайн документ.

Из объектов, содержащих новые цены, инженер может, используя JavaServer Pages ™ (JSP ™), создать HTML – форму для заказа кофе он-лайн. Инженер получает объекты из JSP страницы и включает название и цену каждого из сортов в HTML таблицу на JSP странице. Покупатель водит количество нужного сорта кофе, и нажимает Submit, посылая заказ.

Заключение.

Хотя, для краткости данный сценарий упрощен, он иллюстрирует, как глубоко XML технологии проникают в мир сетевых служб. И теперь, с появлением Java API для XML и J2EE платформы, становится значительно проще использовать Web-службы и писать для них программы.

Словарь специальных терминов (глоссарий)

Asynchronous Асинхронный, слабо связанный, происходящий в разное время. При асинхронной передаче сообщений, отсылается сообщение, а ответ приходит позже при отдельной операции. См. synchronous.

B2B (Business-to-Business) Термин, используемый для описания сетевых взаимодействий между двумя организациями, например, оптовым поставщиком и розничным магазином.

B2C (Business-to-customer) Термин, используемый для описания сетевых взаимодействий между организацией и конечным потребителем, например, между магазином и розничным покупателем.

DOM (Document Object Model) Стандартный API для анализа XML данных, для представления их в виде дерева объектов и обработки содержимого этого дерева. Эта модель разработана консорциумом W3C. JAXP предоставляет интерфейс программирования на языке Java для API этой модели и позволяет приложению включать совместимый DOM анализатор.

DTD (Document Type Definition) Формальное определение шаблона, описывающего вид информации в отдельном XML - документе.

ebXML (Electronic Business XML) XML для электронного бизнеса. Открытая общественная инициатива, которая разрабатывает спецификации, нацеленные на создание единого глобального электронного рынка, основанного на использовании XML и Internet.

HTML (HyperText Markup Language) Язык разметки, используемый для форматирования Web страниц.

HTTP (HyperText Transfer Protocol) Протокол передачи данных в Internet.

J2EE™ (Java™ 2 Platform, Enterprise Edition) Java 2, редакция для предприятий. Инструментальный комплекс Java, определяющий стандарты многоуровневой обработки данных предприятия. Комплекс J2EE включает платформу J2SE.

J2SE™ (Java™ 2 Platform, Standard Edition) Java 2, стандартная поставка. Инструментальный комплекс Java для клиентских вычислений.

JAX Pack Развивающийся набор API языка Java, связанный с XML (JAXP, JAXB, JAXM, JAXR и JAX-RPC). Этот набор будет включен в набор приложений Web Services Pack.

JAXB (Java™ Architecture for XML Binding) Архитектура для преобразования данных в XML документе в объекты языка программирования Java. Задавая шаблон XML документа (например, DTD), компилятор JAXB создаст классы, соответствующие этому DTD. Получившиеся классы содержат средства, позволяющие строить дерево объектов из XML данных, исходя из DTD, и формировать из этого дерева новый XML документ.

JAXM (Java™ API for XML Messaging) Стандартный API для отправки SOAP сообщений с применением языка Java. JAXM базируется на протоколе SOAP 1.1 в спецификации Attachments и предоставляет возможность использовать другие профили, например, ebXML или BizTalk, поверх себя.

JAXP (Java™ API for XML Messaging) Комплексный API для анализа и обработки XML документов**.** JAXP включает поддержку SAX, DOM, XSLT и пространства имен XML (XML Namespaces).

JAXR (Java™ API for XML Registries) Стандартный API для удобного доступа из среды Java к регистрационным формам предприятий в Internet.

JAX-RPC (Java™ API for XML RPC) Стандартный API для удаленного вызова процедур, основанных на XML с использованием языка программирования Java.

loosely coupled (слабосвязанный) Термин, относящийся к взаимодействие между двумя предприятиями, при котором каждое из них не осведомлено об информационной структуры другого предприятия и не зависят от нее.

OASIS (Organization for the Advancement of Structured Information Standards) Международная общественная организация, способствующая принятию взаимодействующих спецификаций открытых стандартов, например XML.

Registry Web-служба, которая дает возможность для активного и свободного сотрудничества между организациями с помощью предоставления доступа к совместной информации. Эту службу иногда сравнивают с "желтыми страницами" электронной коммерции. См. repository

Rerository – Средство хранения данных, во многом схожее с базой данных. Деловая регистрация использует репозиторий для хранения своих данных, например, информации о деятельности, XML описаний определенных протоколов (например, RosettaNet PIP3A4 для заказов) и схем XML, определяющих структуру XML документов, полученных при обмене в течение поддерживаемого делового процесса.

schema Спецификация структуры множества XML документов. Пример: DTD.

SAX Simple API for XML Parsing Стандартный API, задающий основанный на событиях синтаксический анализатор XML. SAX разработан членами XML-DEV mail list, и в данный момент его развитием занимается OASIS standards body. JAXP предоставляет программируемый интерфейс Java для SAX API и позволяет приложениям подключать соответствующий анализатор SAX.

synchronous Тесно связанный, происходящий одновременно. Например, синхронизированные сообщения, после отправления подобного сообщения, ничего не будет происходить до получения ответа. Другими словами, сообщение и ответ тесно связаны. В JAXM API для отправки синхронных сообщений используется метод call. Он посылает сообщение и осуществляет блокировку вплоть до получения ответа.

SOAP (**Simple Object Access Protocol**) **Простой протокол доступа к объектам.** Основанный на XML протокол для доступа к сетевым сервисам и для их интеграции друг с другом.

UDDI (**Universal Description, Discovery, and Integration**) **Универсальное описание, поиск и взаимодействие.** Основанный на языке XML всемирный регистр Web-сервисов электронной коммерции.

web services Интернет службы. Сетевые службы распространяются в Интернете в произвольно связанном виде, используя XML интерфейс. Например, сервисы, основанные на JAX-RPC, представляют собой набор процедур, который можно вызвать с помощью удаленного клиента.

Web Service Pack Набор приложений, который пересылает ключевые технологии для упрощения построения сетевых служб при помощи платформы Java 2. Он включает в себя JavaServer Faces (стандартный API для создания Java Web GUIs), Tomcat (an open-source implementation of JavaServer Pages и Java Servlet technologies), и JAX Pack (узел Java APIs для XML).

WSDL (**Web Services Description Language**) - **Язык описания Web-сервисов.** Основанный на XML язык, служащий для описания доступных через Интернет Web-сервисов.

WWW (**World Wide Web**) - **Всемирная паутина**. Сеть систем и данных, называемая Интернет.

W3C (**World Wide Web Consortium**) **WWW-консорциум.** Организация, создающая и развивающая стандарты Web технологий для обеспечения возможности взаимодействия Web-языков и протоколов.

XML (**eXtensible Markup Language**) - **Расширяемый язык разметки, язык XML.** Простой расширяемый язык для разметки текстовой информации. Благодаря его переносимости служит как универсальный базовый формат для обмена данными между приложениями, распределенных Web-служб и торговыми партнерами.

XML Namespaces Стандарт W3C для построения документов, связанных с несколькими DTD и содержащими совпадающие имена. Стандарт поддерживается JAXP-ом.

XSL (**eXtensible Stylesheet Language**) **Расширяемый язык таблиц стилей, язык XSL.** Язык XSL служит для описания преобразований XML-документов. Для преобразования XML документа, используя таблицу стилей приложения, применяют XSLT, являющийся расширением XSL.

XSLT (**XSL Transformations**) - **XSL-преобразования, стандарт XSLT.** Составная часть стандарта XSL, описывающая язык для преобразования XML-документов в другие форматы, такие как HTML. Для

преобразования XML документа приложения могут применять XSLT, используя таблицу стилей XSL, однако XSLT может применяться и независимо от XSL.