EPAM Systems, RD Dep.

Конспект и раздаточный материал

JAVA.SE.13 Java and XML Programming

REVISION HISTORY					
Ver.	**Description of Change**	**Author**	**Date**	**Approved**	
				Name	**Effective Date**
<1.0>	Первая версия	Игорь Блинов	<06.09.2011>		
<2.0>	Вторая версия. Конспект переделан под обновленное содержание материала модуля.	Ольга Смолякова	<11.06.2014>		

Содержание

Теги, элементы, атрибуты

XML или **Extensible Markup Language** (Расширяемый Язык Разметки), является языком разметки, который можно использовать для создания ваших собственных тегов.

```
<note>
      <to>Вася</to>
      <from>Света</from>
      <heading>Напоминание</heading>
      <body>Позвони мне завтра!</body>
</note>
```

Тег - это текст между левой угловой скобкой (<) и правой угловой скобкой (>). Есть начальные теги (такие, как <name>) и конечные теги (такие, как </name>)

```
<from>                    </heading>
```

Элементом является начальный тег, конечный тег и все, что есть между ними. В примере элемент <name> содержит два дочерних элемента: <title>, <first-name> и <last-name>.

```
<note>
  <to>Вася</to>
  <from>Света</from>
</note>
```

Атрибут - это пара имя-значение внутри начального тега элемента.

```
<note id="1">
```

Правила XML-документа

Корневой элемент

Документ XML должен содержаться в единственном элементе. Этот единственный элемент называется корневым элементом и содержит весь текст и любые другие элементы документа.

```
<?xml version="1.0" encoding="UTF-8"?>
<notes>
      <note id="1">
            <to>Вася</to>
            <from>Света</from>
            <heading>Напоминание</heading>
            <body>Позвони мне завтра!</body>
      </note>
</notes>
```

Элементы не могут перекрываться

Элементы XML не могут перекрывать друг друга.

```
<to>Вася</to>
<from><i>Света</from>
<heading>Напоминание</heading></i>
<body>Позвони мне завтра!</body>
...
```

Конечные теги являются обязательными
Нельзя опускать какие-либо закрывающие теги.

```
...
<to>Вася</to>
<from>Света</from>
<heading><p>Напоминание</heading>
<body><br/>Позвони мне завтра!</body>
...
```

Элементы чувствительны к регистру
Элементы XML чувствительны к регистру.

```
...
<heading>Напоминание</heading>
<body>Позвони мне завтра!</BODY>
...
```

Атрибуты должны иметь значения в кавычках
Есть два правила для атрибутов в XML-документах:

- Атрибуты должны иметь значения
- Эти значения должны быть заключены в кавычки

```
<note id="1">
```

Можно использовать одинарные или двойные кавычки, но только согласованно.

Если значение атрибута содержит одинарные или двойные кавычки, можно использовать другой вид кавычек

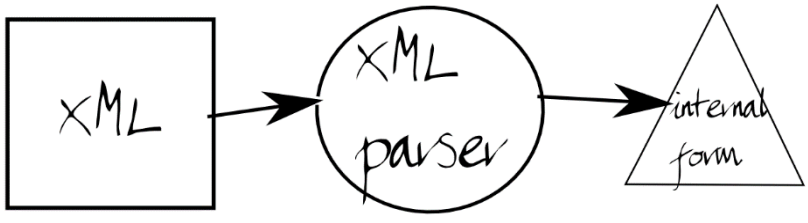
name="Doug's car"

Также допускается сущности *"* для двойной кавычки и *&apos*; для одинарной.

Комментарии

Комментарии могут появляться где угодно в документе; даже перед корневым элементом. Комментарий начинается с <!-- и заканчивается -->. Комментарий не может содержать двойного дефиса (--) нигде, кроме как в конце; за этим исключением, комментарий может содержать что угодно. Любая разметка внутри комментария игнорируется.

```
<!-- комментарий -->
```



Спецификация XML требует, чтобы парсер браковал любой XML-документ, который не выдерживает основные правила.

Парсер - это часть кода, которая пытается прочесть документ и интерпретировать его содержимое.

Есть три вида XML-документов:

- **Неправильные документы** не следуют синтаксическим правилам, определенным спецификациею XML. Если разработчик определил правила для документа, которые могут содержаться в DTD или в схеме, и документ не следует этим правилам, такой документ также является неправильным
- **Правильные документы** следуют синтаксическим правилам XML и правилам, определенным в их DTD или в схеме.
- **Правильно-форматированные документы** следуют синтаксическим правилам XML, но не имеют DTD или в схемы.

Объявления XML

Большинство XML-документов начинаются с XML-объявления, которое обеспечивает базовую информацию о документе для парсера.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Употребление XML-объявления рекомендуется, но не является обязательным. Если оно есть, оно должно быть первым, что есть в документе.

Объявление может содержать до трех пар имя-значение (многие называют их атрибутами, хотя технически они таковыми не являются):

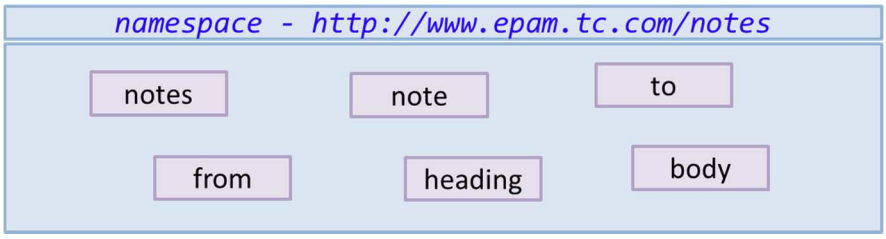
- **version** - используемая версия XML;
- **encoding** - набор символов, используемый в этом документе; если encoding не указан, XML-парсер предполагает набор UTF-8;
- **standalone** - может быть либо yes, либо no, определяет, может ли этот документ быть обработан без чтения каких-либо других файлов.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Пространства имен

Пространство имён (namespace) - это логическая группа уникальных идентификаторов.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tc:notes xmlns:tc="http://www.epam.tc.com/notes"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.epam.tc.com/notes notes.xsd">

      <note id="1">
        <to>Вася</to>
        <from>Света</from>
        <heading>Напоминание</heading>
        <body>Позвони мне завтра!</body>
      </note>
</tc:notes>
```



Чтобы **использовать** пространство имен, необходимо определить префикс пространства имен и отобразить его на определенную строку. Префикс пространства имен уникален в пределах данного документа. Такое ограниченное имя (**qualified name**) однозначно идентифицирует элемент или атрибут и указывает, к какому пространству имен он относится.

```
<readers xmlns:address="http://www.xyz.com/addresses/"
       xmlns:books="http://www.zyx.com/books/"
       xmlns:reader="http://www.zyx.com/readers">

       <reader:book-order>
              <address:full-address>
                     <title>Mrs.</title>
                     <name>Ivales</name>
              </address:full-address>
              <books:title>Lord of the Rings</books:title>
       </reader:book-order>
</readers>
```

Строка в определении пространства имен является только строкой.
Только одно важно в отношении строки пространства имен: она должна быть уникальной.

```
<readers xmlns:address="http://www.xyz.com/addresses/"
       xmlns:books="http://www.zyx.com/books/"
       xmlns:reader="http://www.zyx.com/readers">
```

Определение пространства имен для определенного элемента означает, что все его дочерние элементы принадлежат к тому же пространству имен.

```
<readers xmlns:address="http://www.xyz.com/addresses/"
       xmlns:books="http://www.zyx.com/books/"
       xmlns:reader="http://www.zyx.com/readers">

       <reader:book-order>
              <address:full-address>
                     <title>Mrs.</title>
                     <name>Ivales</name>
              </address:full-address>
              <books:title>Lord of the Rings</books:title>
       </reader:book-order>
</readers>
```

Пространство имен действует только в пределах того элемента, атрибутом которого является его декларация.

```
<readers xmlns:reader="http://www.zyx.com/readers" >
```

```
  <reader:book-order>
    <address:full-address xmlns:address="http://www.xyz.com/addresses/">
                    <title>Mrs.</title>
                    <name>Ivales</name>
    </address:full-address>
    <books:title xmlns:books="http://www.zyx.com/books/">
       Lord of the Rings
    </books:title>
  </reader:book-order>
</readers>
```

При использовании пространств имен важно учитывать, что атрибуты элемента не наследуют его пространство имен. Иными словами, если префикс пространства имен для атрибута не указан, то его имя не относится ни к какому пространству имен.

Существует два способа декларации пространства имен: декларация по умолчанию и явная декларация.

Декларация по умолчанию объявляет пространство имен для всех элементов и их атрибутов, которые содержатся в данном элементе.	Явная декларация
xmlns=URI	**xmlns:имя=URI**

`<readers`
`xmlns="http://www.zyx.com/readers"`
 	
`xmlns:address="http://www.xyz.com/addresses" >`
 	
`<reader:book-order>`
`<address:full-address>`
`<title>Mrs.</title>`
`<name>Ivales</name>`
`</address:full-address>`
`</reader:book-order>`
`</readers>`
	`<readers`
	`xmlns:reader="http://www.zyx.com/readers"`
	
	`xmlns:address="http://www.xyz.com/addresses"`
	`>`
	
	`<reader:book-order>`
	`<address:full-address>`
	`<title>Mrs.</title>`
	`<name>Ivales</name>`
	`</address:full-address>`
	`</reader:book-order>`
	`</readers>`

Префикс xml не требует декларации. Он зарезервирован для расширений языка XML и всегда относится к пространству имен

"http://www.w3.org/XML/1998/namespace".

С его помощью можно, например, задать базовый URI любого элемента с помощью атрибута xml:base следующего вида:

xml:base=URI

```
<readers xmlns:reader="http://www.zyx.com/readers"
       xmlns:book="http://www.zyx.com/books/"
       xml:base="http://www.company.com/readers/">

       <reader:book-order>

         <book:title>Lord of the Rings</book:title>
         <book:list-of-pictures xml:base="/pictures/">
```

```
            <book:book-picture xml:base="picture1.jpg">Рисунок 1
            </book:book-picture>
            <book:book-picture xml:base="picture2.jpg">Рисунок 2
            </book:book-picture>
        </book:list-of-pictures>
      </reader:book-order>
</readers>
```

XSD

Схема XSD представляет собой строгое описание XML-документа. XSD-схема является XML-документом.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.epam.tc.com/note"
xmlns:tns="http://www.epam.tc.com/note">
<element name="notes">
  <complexType>
    <sequence>
        <element name="note" type="tns:Note"
             minOccurs="1"
             maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
<complexType name="Note">
  <sequence>
    <element name="to" type="string" />
    <element name="from" type="string" />
    <element name="heading" type="string" />
    <element name="body" type="string" />
  </sequence>
  <attribute name="id" type="int" use="required" />
</complexType>
</schema>
```

С помощью схемы XSD можно также проверить документ на корректность.

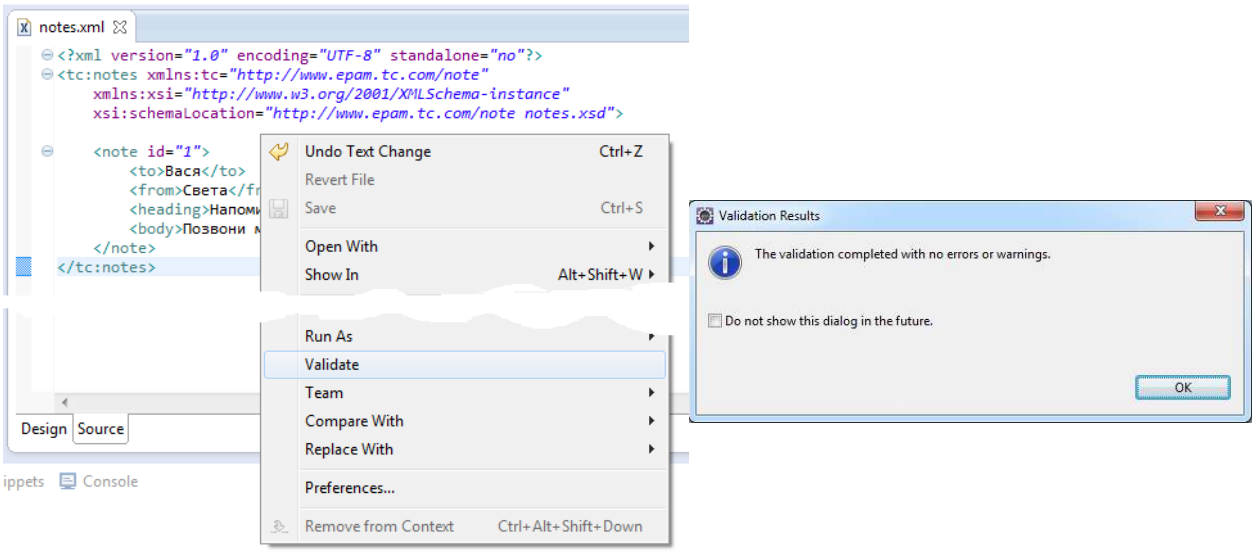


Схема XSD содержит 44 базовых типа и имеет поддержку пространств имен (**namespace**).

Built-in type	Example	Built-in type	Example
string	This is a string	**int**	-1, 126789675
normalizedString	This is a string	**unsignedInt**	0, 1267896754
token	This is a token	**long**	-1, 12678967543233
byte	-1, 126	**unsignedLong**	0, 12678967543233
unsignedByte	0, 126	**short**	-1, 12678
base64Binary	GpM7	**unsignedShort**	0, 12678
hexBinary	0FB7	**decimal**	-1.23, 0, 123.4, 1000.00
integer	-126789, -1, 0, 1, 126789	**float**	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
positiveInteger	1, 126789	**double**	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
negative	-126789, -1	**boolean**	true, false 1, 0
nonNegativeInteger	0, 1, 126789	**time**	13:20:00.000, 13:20:00.000-05:00
nonPositiveInteger	-126789, -1, 0	**dateTime**	1999-05-31T13:20:00.000-05:00
duration	P1Y2M3DT10 H30M12.3S	**anyURI**	http://www.example.com/,http://www.example.com/ doc.html#ID5
date	1999-05-31	**language**	en-GB, en-US, fr
gMonth	--05--	**ID**	*XML 1.0 атрибут типа ID*
gYear	1999	**IDREF**	*XML 1.0 атрибут типа IDREF*
gYearMonth	1999-02	**IDREFS**	*XML 1.0 атрибут типа IDREFS*
gDay	---31	**ENTITY**	*XML 1.0 атрибут типа ENTITY*
gMonthDay	--05-3	**ENTITIES**	*XML 1.0 атрибут типа ENTITIES*
Name	*XML 1.0 тип Name*	**NOTATION**	*XML 1.0 атрибут типа NOTATION*
QName	po:USAddress	**NMTOKEN**	*XML 1.0 атрибут типа NMTOKEN*
NCName	USAddress	**NMTOKENS**	*XML 1.0 атрибут типа NMTOKENS*

Правила описания xsd-схемы.

<schema> - корневой элемент XML-схемы.

```
<schema>
</schema>

<schema
       xmlns:xs="http//www.w3.org/2001/XMLSchema"      xmlns="http://www.epamrd.com"
       targetNamespace="http://www.epamrd.com"         elementFormDefault="qualified"
       attributeFormDefault="qualified">
</schema>
```

<element> - основные блоки XML-документа, содержащие данные и определяющие структуру описываемого документа.

```
<element name="x" type="y"/>

<element name="customer-name" type="string" default="unknown"/>
<element name="customer-location" type="string" fixed=" UK"/>
```

Cardinality: minOccurs, maxOccurs – ограничения, накладываемые на определенные числовые элементы (по умолчанию 1).

```
<element name="Customer_order" type="integer"
             minOccurs ="0"  maxOccurs="unbounded"/>

<element name="Customer_hobbies" type="string"
             minOccurs="2"  maxOccurs="10"/>
```

Simple Types (простые типы) – наследуются от встроенных типов (string, integer) и позволяют создавать собственные типы данных.

Extending Simple Types. Restriction –позволяет ограничить имеющиеся простые типы.

```
<simpleType name="LetterType">
       <restriction base="string">
              <pattern value="[a-zA-Z]"/>
       </restriction>
</simpleType>
```

Ограничения могут использовать так называемые грани (Facets).

Facet	Описание
<minLength value="3"> <maxLength value="8">	Ограничение длины значений типа
<minInclusive value="0"> <maxInclusive value="10">	Ограничение значений типа
<xs:length value="30">	Ограничение длины типа

`<enumeration value="Hippo"/>`
`<enumeration value="Zebra"/>`
`<enumeration value="Lion"/>`
	Задание значений типа в виде перечисления

`<pattern value="[0-9]"/>`	Задания ограничения значений типа в виде паттерна

Полный список возможных facets можно посмотреть в стандарте http://www.w3.org/TR/xmlschema-2/#rf-facets.

Extending Simple Types. Union – механизм для объединения двух или более различных типов данных в один.

```
<simpleType name="SizeByNumberType">
	<restriction base="positiveInteger">
		<maxInclusive value="21"/>
	</restriction>
</simpleType>

<simpleType name="SizeByStringNameType">
	<restriction base="string">
		<enumeration value="small"/>
		<enumeration value="medium"/>
		<enumeration value="large"/>
	</restriction>
</simpleType>

<simpleType name="USClothingSizeType">
	<union memberTypes="SizeByNumberType SizeByStringNameType" />
</simpleType>
```

Extending Simple Types. List – позволяет указывать в XML ряд допустимых значений, разделенных пробелами.

```
<simpleType name="SizesinStockType">
	<list itemType="SizeByNumberType" />
</simpleType>

<xs:element name="WinningNumbers" maxOccurs="unbounded">
  <xs:simpleType>
    <xs:list>
      <xs:simpleType>
        <xs:restriction base="xs:unsignedByte">
       <xs:minInclusive value="1" />
       <xs:maxInclusive value="49" />
        </xs:restriction>
      </xs:simpleType>
    </xs:list>
  </xs:simpleType>
</xs:element>

<!-- Valid "WinningNumbers" list values -->
<WinningNumbers>  1 20 24 33 37 43 </WinningNumbers>
```

Complex Types (сложные типы) – это контейнеры для определения элементов, они позволяют определять дочерние элементы для других элементов.

```
<element name="Customer">
    <complexType>
        <sequence>
            <element name="Dob" type="date" />
            <element name="Address" type="string" />
        </sequence>
    </complexType>
</element>
```

Compositors – определяет правила описания дочерних элементов в родительском в документе XML.

sequence	Дочерние элементы, описываемые xsd-схемой, могут появляться в XML-документе только в указанном порядке.
choice	Только один из дочерних элементов, описываемых xsd-схемой, может появиться в XML-документе.
all	Дочерние элементы, описываемые xsd-схемой, могут появляться в XML-документе в любом порядке.

Global Types – сложные типы данных можно объявить не только внутри элемента, но и вне его.

```
<complexType name="AddressType">
    <sequence>
        <element name="Line1" type="string"/>
        <element name="Line2" type="string"/>
    </sequence>
</complexType>

<element name="Customer">
    <complexType>
        <sequence>
            <element name="Dob" type="date"/>
            <element name="Address" type="xs:AddressType"/>
        </sequence>
    </complexType>
</element>
```

Attributes – атрибуты предоставляют дополнительную информацию в пределах элемента.

```
<attribute name="x" type="y"/>

<attribute name="ID" type="string"/>
<attribute name="ID" type="string" use="optional"/>
<attribute name="ID" type="string" use=" prohibited"/>
```

Mixed Content – позволяет смешивать элементы и данные.

```
<element   name="MarkedUpDesc">
    <complexType   mixed="true">
        <sequence>
       <element   name="Bold"   type="string"   />
       <element   name="Italic"   type="string"   />
        </sequence>
    </complexType>
</element>

<MarkedUpDesc>
    This is an
    <Bold>Example</Bold>
    of
    <Italic>Mixed</Italic>
    Content,
    Note there are elements mixed in with the elements data.
</MarkedUpDesc>
```

<group> и **<attributeGroup>** - объединяют элементы(атрибуты) в группы, позволяя на них ссылаться. Такие группы не могут быть расширены или ограничены.

```
<group name="CustomerDataGroup">
  <sequence>
    <element name="Forename" type="string" />
    <element name="Surname" type="string" />
    <element name="Dob" type="date" />
  </sequence>
</group>
<attributeGroup name="DobPropertiesGroup">
  <attribute name="Day" type="string" />
  <attribute name="Month" type="string" />
  <attribute name="Year" type="integer" />
</attributeGroup>

<complexType name="Customer">
    <sequence>
        <group ref="u:CustomerDataGroup"/>
        <element name="..." type="..."/>
    </sequence>
    <attributeGroup ref="u:DobPropertiesGroup"/>
</complexType>
```

<any> - определяет, что документ может содержать элементы, неопределенные в XML-схеме.

```
<element name="Message">
  <complexType>
    <sequence>
      <element name="DateSent" type="date" />
      <element name="Sender" type="string" />
      <element name="Content">
        <complexType>
          <sequence>
            <any />
          </sequence>
        </complexType>
      </element>
    </sequence>
```

```
  </complexType>
</element>

<Message>
 <DateSent>2000-01-12</DateSent>
 <Sender>Admin</Sender>
 <Content>
  <AccountCreationRequest>
   <AccountName>Fred</AccountName>
  </AccountCreationRequest>
 </Content>
</Message>
```

<anyAttribute> - позволяет указывать в элементе атрибут, не определенный в XML-схеме.

```
<element name="Sender">
  <complexType>
    <simpleContent>
      <extension base="string">
        <anyAttribute />
      </extension>
    </simpleContent>
  </complexType>
</element>

<Sender ID="7687">Fred</Sender>
```

Квалификация

Квалификация необходима для однозначного разграничения пространств имен элементов и атрибутов.

Для того, чтобы все локально объявленные элементы в были квалифицированы, необходимо установить значение **elementFormDefault** (<scheme>) равным **qualified**.

Атрибуты, которые должны быть квалифицированы (либо потому что объявлены глобально, либо потому что признак **attributeFormDefault**, установлен в **qualified**), в документах появляются с префиксом.

```
<xs:schema
       xmlns:xs="http//www.w3.org/2001/XMLSchema"
       xmlns="http://www.epam.com"
       targetNamespace="http://www.epam.com"
       elementFormDefault="qualified"
       attributeFormDefault="qualified">
</schema>
```

Card.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.epamrd.org/card"
        xmlns:tns="http://www.epamrd.org/card"
        elementFormDefault="qualified">


      <complexType name="CardType">
            <sequence>
                  <element name="message" type="string" />
```

```
                    <element name="color" type="string" />
                </sequence>
        </complexType>

</schema>
```

Sock.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.epamrd.org/sock"
        xmlns:tns="http://www.epamrd.org/sock"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">

       <complexType name="SockType">
              <sequence>
                     <element name="color" type="string"/>
              </sequence>
              <attribute name="size" type="integer" />
       </complexType>

</schema>
```

Presents.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.epamrd.org/Presents"
        xmlns:tns="http://www.epamrd.org/Presents"
        xmlns:s="http://www.epamrd.org/sock"
        xmlns:c="http://www.epamrd.org/card"
        elementFormDefault="qualified">
  <import schemaLocation="Sock.xsd" namespace="http://www.epamrd.org/sock" />
  <import schemaLocation="Card.xsd" namespace="http://www.epamrd.org/card" />

  <element name="presents" type="tns:PresentType" />

  <complexType name="PresentType">
    <sequence>
      <element name="present">
        <complexType>
          <sequence>
       <element name="sock" type="s:SockType" />
       <element name="card" type="c:CardType" />
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

Presents.xml

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

<pr:presents xmlns:pr="http://www.epamrd.org/Presents"
   xmlns:s="http://www.epamrd.org/sock"
   xmlns:c="http://www.epamrd.org/card"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.epamrd.org/Presents Presents.xsd">

       <pr:present>
              <pr:sock s:size="4">
```

```
                <s:color>red</s:color>
            </pr:sock>
            <pr:card>
                <c:message>Happy New Year!</c:message>
                <c:color>gold</c:color>
            </pr:card>
        </pr:present>

</pr:presents>
```

DTD

Для описания структуры XML-документа используется язык описания **DTD (Document Type Definition)**.

DTD определяет, какие теги (элементы) могут использоваться в XML-документе, как эти элементы связаны между собой (например, указывать на то, что элемент **<student>** включает дочерние элементы **<name>**, **<telephone>** и **<address>**), какие атрибуты имеет тот или иной элемент.

Подключение DTD

1. **<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>**
   <! DOCTYPE students SYSTEM "students.dtd">

2. **<?xml version="1.0" ?>**
   <! DOCTYPE student
   [<!ELEMENT student (name, telephone, address)>*<!--далее идет описание элементов* name, telephone, address *-->***]**
   >

students.xml

```
<?xml version="1.0" ?>
<!DOCTYPE students SYSTEM "students.dtd" >
<students>
  <student login="mit" faculty="mmf">
    <name>Mitar Alex</name>
    <telephone>2456474</telephone>
    <address>
      <country>Belarus</country>
      <city>Minsk</city>
      <street>Kalinovsky 45</street>
    </address>
  </student>
  <student login="pus" faculty="mmf">
    <name>Pashkun Alex</name>
      <telephone>3453789</telephone>
      <address>
        <country>Belarus</country>
        <city>Brest</city>
        <street>Knorina 56</street>
        </address>
  </student>
</students>
```

students.dtd

```
<!ELEMENT students (student)+>
<!ELEMENT student (name, telephone, address)>
```

```
<!ATTLIST student
       login ID #REQUIRED
       faculty CDATA #REQUIRED
>
<!ELEMENT name (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT address (country, city, street)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT street (#PCDATA)>
```

Описание элемента

<!ELEMENT name (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT address (country, city, street)>

PCDATA. - элементы могут содержать любую информацию, с которой может работать программа-анализатор (**PCDATA** – parsed character data). Есть также маркеры **EMPTY** – элемент пуст и **ANY** – содержимое специально не описывается.

Если в определении элемента указывается "смешанное" содержимое, т.е. текстовые данные или набор элементов, то необходимо сначала указать PCDATA, а затем разделенный символом "|" список элементов.

Для того, чтобы указать количество повторений включений элементов могут использоваться символы: ‘**+**’(один или много), ‘*****’(0 или много), ‘**?**’(0 или 1)

- **<!ELEMENT student (name, telephone, address)>** - элемент **student** содержит один и только один элемент **name**, **telephone** и **address**.

Если существует несколько вариантов содержимого элементов, то используется символ ‘|’ (или).

- **<!ELEMENT student (#PCDATA | body)> -** элемент **student** может содержать либо дочерний элемент **body**, либо **PCDATA**.
- **<!ELEMENT issue (title, author+, table-of-contents?)>** - внутри.
- **<!ELEMENT flower (PCDATA | title)*>**

Описание атрибутов

<!ATTLIST название_елемента название_атрибута тип_атрибута значение_по_умолчанию >

Например:

<!ATTLIST student
login ID #REQUIRED
faculty CDATA #REQUIRED>

Существует несколько возможных значений атрибута, это:

- **CDATA** – значением атрибута является любая последовательность символов;
- **ID** – определяет уникальный идентификатор элемента в документе;

- **IDREF** (**IDREFS**) – значением атрибута будет идентификатор (список идентификаторов), определенный в документе;
- **ENTITY** (**ENTITIES**) – содержит имя внешней сущности (несколько имен, разделенных запятыми);
- **NMTOKEN** (**NMTOKENS**) – слово (несколько слов, разделенных пробелами).

Опционально можно задать значение по умолчанию для каждого атрибута. Значения по умолчанию могут быть следующими:

- **#REQUIRED** – означает, что атрибут должен присутствовать в элементе;
- **#IMPLIED** – означает, что атрибут может отсутствовать, и если указано значение по умолчанию, то анализатор подставит его.
- **#FIXED defaultValue** – означает, что атрибут может принимать лишь одно значение, то, которое указано в DTD.

defaultValue – значение по умолчанию, устанавливаемое парсером при отсутствии атрибута. Если атрибут имеет параметр **#FIXED**, то за ним должно следовать **defaultValue**.

Если в документе атрибуту не будет присвоено никакого значения, то его значение будет равно заданному в DTD.

defaultValue – значение по умолчанию, устанавливаемое парсером при отсутствии атрибута. Если атрибут имеет параметр **#FIXED**, то за ним должно следовать **defaultValue**.

Если в документе атрибуту не будет присвоено никакого значения, то его значение будет равно заданному в DTD. Значение атрибута всегда должно указываться в кавычках.

Определение сущности

Сущность (entity) представляет собой некоторое определение, чье содержимое может быть повторно использовано в документе. Описывается сущность с помощью дескриптора **!ENTITY**:

```
<!ENTITY company 'Sun Microsystems'>
<sender>&company;</sender>
```

Программа-анализатор, которая будет обрабатывать файл, автоматически подставит значение Sun Microsystems вместо **&company**.

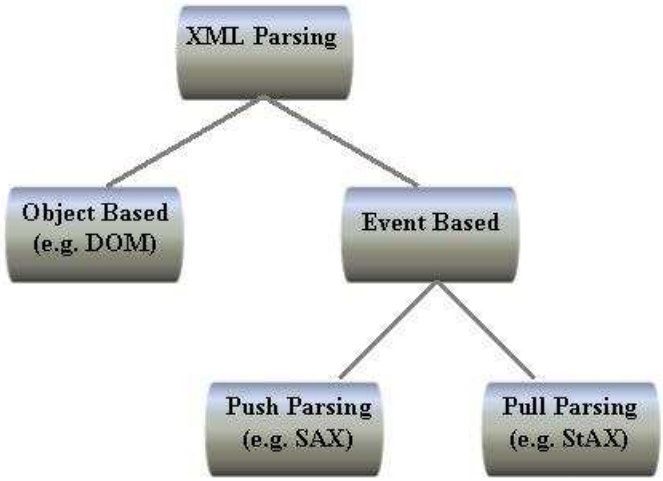
Для повторного использования содержимого внутри описания DTD используются параметрические (параметризованные) сущности.

```
<!ENTITY  % elementGroup "firstName, lastName,gender, address, phone">
<!ELEMENT employee (%elementGroup)>
<!ELEMENT contact (%elementGroup)>
```

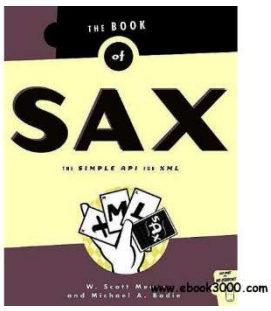
В XML включены внутренние определения для символов. Кроме этого, есть внешние определения, которые позволяют включать содержимое внешнего файла:

```
<!ENTITY logotype SYSTEM "/image.gif" NDATA GIF87A>
```

XML PARSERS



SAX



SAX - это событийный парсер для XML, т.е. он последовательно читает и разбирает данные из входного потока (это может быть файл, сетевое соединение, или любой другой InputStream).

Когда парсер находит структурный элемент (открывающий тег, закрывающий тег, и т.п.), он оповещает об этом слушателя (обработчик события), и передает ему в качестве параметра найденный элемент.

SAX делает возможным привязку специфичного для приложения кода к событиям.

SAX (SAX2) определяет интерфейс,

org.xml.sax.XMLReader

который должны реализовывать все SAX-совместимые анализаторы XML. (Благодаря этому SAX точно знает, какие методы доступны для обратного вызова и использования в приложении).

```
// создание экземпляра класса Reader
org.xml.sax.XMLReader reader =
            XMLReaderFactory.createXMLReader();
// делаем что-то с помощью анализатора
reader.parse(url);
```

Для анализа документа применяется метод **parse()** класса org.xml.sax.XMLReader.

У качестве параметра может выступать экземпляр класса **org.xml.sax.InputSource**, либо строка, содержащая URI.

```
// создание экземпляра класса Reader
```

```
org.xml.sax.XMLReader reader =
  XMLReaderFactory.createXMLReader(vendorParserClass);
// регистрируем обработчик содержимого
// регистрируем обработчик ошибок
// анализируем
InputSource inputSource = new InputSource(xmlURI);
reader.parse(inputSource);
```

Используя **InputSource** и заключив в него переданный URI можно определить системный идентификатор документа. По сути дела, устанавливается путь к документу для анализатора, что и позволяет разрешать все относительные пути внутри этого документа.

```
InputSource inputSource =
      new InputSource(
            new java.io.FileInputStream(
                  new java.io.File(xmlURI)));
```

Handler

Обработчик содержимого - это набор методов обратного вызова SAX, позволяющих программистам связывать код приложения с событиями, возникающими во время синтаксического разбора документа.

SAX обрабатывает документ последовательно и не загружает его в память целиком.

В SAX 2.0 определены четыре основных интерфейса-обработчика:

- **org.xml.sax.ContentHandler** –обработчик событий документа
- **org.xml.sax.ErrorHandler** – обработки ошибочных ситуаций
- **org.xml.sax.DTDHandler** – обработчик событий при анализе DTD-описаний
- **org.xml.sax.EntityResolver** - обработчик событий загрузки DTD-описаний (создан специально для интерпретации внешних сущностей, на которые ссылается XML-документ)

Классы, реализующие эти интерфейсы можно зарегистрировать в анализаторе с помощью методов **setContentHandler(); setEntityResolver(); setDTDHandler(); setErrorHandler();**

Интерфейс ContentHandler

```
public interface ContentHandler {
  public void setDocumentLocator(Locator locator);
  public void startDocument() throws SAXException;
  public void endDocument() throws SAXException;
  public void startPrefixMapping(String prefix, String uri)
                  throws SAXException;
  public void endPrefixMapping(String prefix) throws SAXException;
  public void startElement(String uri, String localName, String qName,
                  Attributes atts) throws SAXException;
  public void endElement(String uri, String localName, String qName)
                  throws SAXException;
  public void characters(char ch[], int start, int length)
                  throws SAXException;
  public void ignorableWhitespace(char ch[], int start, int length)
                  throws SAXException;
  public void processingInstruction(String target, String data)
                  throws SAXException;
  public void skippedEntity(String name) throws SAXException;
}
```

Методы интерфейса ContentHandler

Метод	Назначение
setDocumentLocator(Locator locator)	Указатель позиции в документе, действителен только для текущего цикла анализа.
startDocument()	Вызывается первым во всем процессе анализа документа, за исключением метода setDocumentLocator()
endDocument()	Вызывается последним, в том числе и среди методов остальных обработчиков SAX
startElement(String uri, String localName, String qName, Attributes atts)	Сообщает о начале анализа документа, предоставляет приложению информацию об элементе XMLи любых его атрибутах
endElement(String uri, String localName, String qName)	Сообщает о достижении закрывающего тега элемента.
startPrefixMapping(String prefix, String uri)	Вызывается перед методом, связанным с элементом, в котором пространство имен объявлено.
endPrefixMapping(String prefix)	Вызывается после закрытия элемента в котором пространство имен объявлено.
characters(**char** ch[], **int** start, **int** length)	Сообщает о достижении символьного значения элемента
ignorableWhitespace(**char** ch[], **int** start, **int** length)	Метод обратного вызова для необрабатываемых символов между элементами; должен вызываться только при наличии DTD или XML схемы, задающих ограничения.
processingInstruction(String target, String data)	Метод обратного вызова для обработки PI-инструкций
skippedEntity(String name)	Метод обратного вызова, который выполняется, если анализатор без проверки действительности пропускает сущность (анализаторы без проверки действительности не обязаны (но могут) интерпретировать ссылки на сущности).

В SAX 2 поддержка пространства имен осуществляется на уровне элементов. Это позволяет различать пространство имен элемента, представленное префиксом элемента и связанным с этим префиксом URI, и локальное имя элемента (имя без префикса).

Область отображения префикса – это элемент с атрибутом xmlns, объявляющий пространство имен.

Интерфейс DocumentHandler

```
public interface DocumentHandler {
  void setDocumentLocator(Locator locator);
  void startDocument() throws SAXException;
  void endDocument() throws SAXException;
  void startElement(String name, AttributeList atts)
                    throws SAXException;
  void endElement(String name) throws SAXException;
  void characters(char ch[], int start, int length)
                    throws SAXException;
  void ignorableWhitespace(char ch[], int start, int length)
                    throws SAXException;
  void processingInstruction(String target, String data)
                    throws SAXException;
}
```

Интерфейс ErrorHandler

```
public interface ErrorHandler {
  public abstract void warning(SAXParseException exception)
                    throws SAXException;
  public abstract void error(SAXParseException exception)
                    throws SAXException;
  public abstract void fatalError(SAXParseException exception)
                    throws SAXException;
}
```

Метод	Назначение
void warning(SAXParseException exception)	Предупреждения
void error(SAXParseException exception)	Некритические ошибки
void fatalError(SAXParseException exception)	Критические ошибки

Интерфейс DTDHandler

```
public interface DTDHandler {
  public abstract void notationDecl(String name, String publicId,
                    String systemId) throws SAXException;
  public abstract void unparsedEntityDecl(String name, String publicId,
       String systemId, String notationName) throws SAXException;
}
```

Интерфейс DTDHandler позволяет получать уведомление, когда анализатор встречает неанализируемую сущность или объявление нотации.

Интерфейс EntityResolver

```
public interface EntityResolver {

  public abstract InputSource resolveEntity(String publicId,
       String systemId)    throws SAXException, IOException;

}
```

Интерфейс интерпретирует сущности. Обычно сущность по публичному или системному идентификатору интерпретируется анализатором XML. Если метод resolveEntity() возвращает null, этот процесс протекает в традиционном варианте. Если вернуть из метода корректный объект InputSource, то вместо указанного публичного или системного идентификатора в качестве значения ссылки на сущности будет использоваться этот объект.

```
public interface Locator {
	public abstract String getPublicId();
	public abstract String getSystemId();
	public abstract int getLineNumber();
	public abstract int getColumnNumber();
}
```

Интерфейс Locator позволяет определить текущую позицию в XML файле. Поскольку эта позиция действительна только для текущего цикла анализа, локатор следует использовать только в области видимости реализации интерфейса ContentHandler.

```
public class DefaultHandler
implements EntityResolver, DTDHandler, ContentHandler, ErrorHandler
{
…
}
```

Класс DefaultHandler реализует интерфейсы ContentHandler, ErrorHandler, EntityResolver, DTDHandler и предоставляет пустые реализации для каждого метода каждого интерфейса.

menu.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast-menu>
	<food id="1">
		<name>Belgian Waffles</name>
		<price>$5.95</price>
		<description>
			two of our famous Belgian Waffles with plenty of real maple syrup
		</description>
		<calories>650</calories>
	</food>
	<food id="2">
		<name>Strawberry Belgian Waffles</name>
		<price>$7.95</price>
		<description>
			light Belgian waffles covered with strawberrys and whipped cream
		</description>
		<calories>900</calories>
	</food>
	<food id="3">
		<name>Berry-Berry Belgian Waffles</name>
		<price>$8.95</price>
		<description>
			light Belgian waffles covered with an assortment of fresh berries
and
			whipped cream
		</description>
		<calories>900</calories>
	</food>
```

```
	<food id="4">
		<name>French Toast</name>
		<price>$4.50</price>
		<description>
			thick slices made from our homemade sourdough bread
		</description>
		<calories>600</calories>
	</food>
	<food id="5">
		<name>Homestyle Breakfast</name>
		<price>$6.95</price>
		<description>
			two eggs, bacon or sausage, toast, and our ever-popular hash
browns
		</description>
		<calories>950</calories>
	</food>
</breakfast-menu>
```

MenuTagName.java

```
package _java._se._13._sax;
public enum MenuTagName {
	NAME, PRICE, DESCRIPTION, CALORIES, FOOD, BREAKFAST_MENU
}
```

MenuSaxHandler.java

```
package _java._se._13._sax;
import java.util.ArrayList;
import java.util.List;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
public class MenuSaxHandler extends DefaultHandler {
	private List<Food> foodList = new ArrayList<Food>();
	private Food food;
	private StringBuilder text;

	public List<Food> getFoodList() {
		return foodList;
	}

	public void startDocument() throws SAXException {
		System.out.println("Parsing started.");
	}

	public void endDocument() throws SAXException {
		System.out.println("Parsing ended.");
	}

	public void startElement(String uri, String localName, String qName,
			Attributes attributes) throws SAXException {
		System.out.println("startElement -> " + "uri: " + uri + ", localName: " +
localName
				+ ", qName: " + qName);

		text = new StringBuilder();
		if (qName.equals("food")){
			food = new Food();
			food.setId((Integer.parseInt(attributes.getValue("id"))));
		}
	}
```

```
	public void characters(char[] buffer, int start, int length) {
		text.append(buffer, start, length);
	}

	public void endElement(String uri, String localName, String qName)
			throws SAXException {
		MenuTagName tagName = MenuTagName.valueOf(qName.toUpperCase().replace("-
", "_"));
		switch(tagName){
		case NAME:
			food.setName(text.toString());
			break;
		case PRICE:
			food.setPrice(text.toString());
			break;
		case DESCRIPTION:
			food.setDescription(text.toString());
			break;
		case CALORIES:
			food.setCalories(Integer.parseInt(text.toString()));
			break;
		case FOOD:
			foodList.add(food);
			food = null;
			break;
		}
	}

    public void warning(SAXParseException exception) {
		System.err.println("WARNING: line " + exception.getLineNumber() + ": "
				+ exception.getMessage());
	}

	public void error(SAXParseException exception) {
		System.err.println("ERROR: line " + exception.getLineNumber() + ": "
				+ exception.getMessage());
	}

	public void fatalError(SAXParseException exception) throws SAXException {
		System.err.println("FATAL: line " + exception.getLineNumber() + ": "
				+ exception.getMessage());
		throw (exception);
	}
}
```

SaxDemo.java

```
package _java._se._13._sax;
import java.io.IOException;
import java.util.List;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;

public class SaxDemo {

	public static void main(String[] args) throws ParserConfigurationException,
			SAXException, IOException {

		XMLReader reader = XMLReaderFactory.createXMLReader();
		MenuSaxHandler handler = new MenuSaxHandler();
```

```
                reader.setContentHandler(handler);
                reader.parse(new InputSource("menu.xml"));

                // включение проверки действительности
                reader.setFeature("http://xml.org/sax/features/validation", true);

                // включение обработки пространств имен
                reader.setFeature("http://xml.org/sax/features/namespaces", true);

                // включение канонизации строк
                reader.setFeature("http://xml.org/sax/features/string-interning", true);

                // отключение обработки схем
                reader.setFeature("http://apache.org/xml/features/validation/schema",
                                false);

                List<Food> menu = handler.getFoodList();

                for (Food food : menu) {
                        System.out.println(food.getName());
                }
        }
}
```

Food.java

```
package _java._se._13._sax;
public class Food {
        private int id;
        private String name;
        private String price;
        private String description;
        private int calories;

        public int getId() {
                return id;
        }

        public void setId(int id) {
                this.id = id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String getPrice() {
                return price;
        }

        public void setPrice(String price) {
                this.price = price;
        }

        public String getDescription() {
                return description;
        }

        public void setDescription(String description) {
                this.description = description;
```

```
        }

        public int getCalories() {
              return calories;
        }

        public void setCalories(int calories) {
              this.calories = calories;
        }

}
```

Расширенные возможности SAX 2

```
public interface XMLReader
{
    public boolean getFeature (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public void setFeature (String name, boolean value)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public Object getProperty (String name)
        throws SAXNotRecognizedException, SAXNotSupportedException;
    public void setProperty (String name, Object value)
        throws SAXNotRecognizedException, SAXNotSupportedException;

    // Event handlers.
}
```

В SAX 2 определен стандартный механизм для установки свойств и возможностей анализатора, что позволяет добавлять новые свойства и возможности, если они утверждены консорциумом W3C, без использования фирменных расширений и методов.

```
// включение проверки действительности
reader.setFeature("http://xml.org/sax/features/validation",  true);

// включение обработки пространств имен
reader.setFeature("http://xml.org/sax/features/namespaces", true);

// включение канонизации строк
reader.setFeature("http://xml.org/sax/features/string-interning", true);

// отключение обработки схем
reader.setFeature("http://apache.org/xml/features/validation/schema", false);
```

На страницах

http://xerces.apache.org/xerces-j/features.html
http://xerces.apache.org/xerces-j/properties.html

перечислены все возможности и свойства, поддерживаемые анализатором Apache Xerces.

org.xml.sax.XMLFilter

```
public interface XMLFilter extends XMLReader
{
    public abstract void setParent (XMLReader parent);
    public abstract XMLReader getParent ();
```

```
}
```

XMLFilter предназначен для создания цепей реализаций XMLReader посредством фильтрации.

org.xml.sax.helpers.XMLFilterImpl

```
public class XMLFilterImpl implements XMLFilter, EntityResolver, DTDHandler,
ContentHandler, ErrorHandler {
	// Construct an empty XML filter, with no parent.
	public XMLFilterImpl() {
		super();
	}
	// Construct an XML filter with the specified parent.
	public XMLFilterImpl(XMLReader parent) {
		super();
		setParent(parent);
	}
}
```

XMLFilterImpl позволяет создать конвейер для всех событий SAX, при этом переопределив любые методы, которые необходимо переопределить для добавления в конвейер поведения, определяемого разработчиком приложения.

XMLFilterImpl передаст в неизменном виде события, для которых не были переопределены методы.

ElementFilter.java

```
package _java._se._13._sax.filter;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.XMLFilterImpl;
public class ElementFilter extends XMLFilterImpl{

	public void startElement(String uri, String localName, String qName,
			Attributes attributes) throws SAXException {
		System.out.println("startElement in ElementFilter");
		super.startElement(uri+"2", localName, qName, attributes);
	}

}
```

NamespaceFilter.java

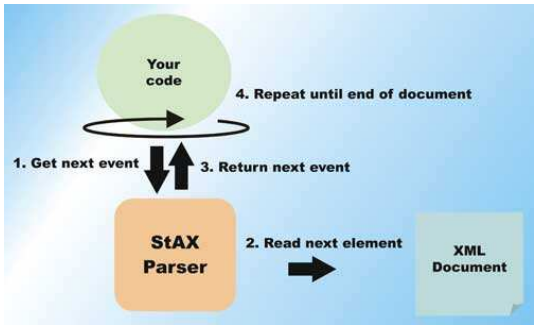
```
package _java._se._13._sax.filter;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLFilterImpl;

public class NamespaceFilter extends XMLFilterImpl {

	public NamespaceFilter(XMLReader reader){
		super(reader);
	}

	public void startPrefixMapping(String prefix, String uri) throws SAXException {
		System.out.println("startPrefixMapping in NamespaceFilter -" + prefix +
", " + uri);
		super.startPrefixMapping(prefix, uri+"2");
	}
```

```
}
```

SAXFilterDemo.java

```
package _java._se._13._sax.filter;
import java.io.IOException;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;
import _java._se._13._sax.MenuSaxHandler;

public class SAXFilterDemo {

      public static void main(String[] args) throws ParserConfigurationException,
SAXException, IOException {
            XMLReader reader = XMLReaderFactory.createXMLReader();
            NamespaceFilter namespaceFilter = new NamespaceFilter(reader);
            ElementFilter elementFilter = new ElementFilter();
            elementFilter.setParent(namespaceFilter);
          MenuSaxHandler handler = new MenuSaxHandler();
          elementFilter.setContentHandler(handler);
          elementFilter.parse(new InputSource("menu.xml"));
      }
}
```

StAX



StAX (Streaming API for XML), который еще называют pull-парсером, включен в JDK, начиная с версии Java SE 6.

Он похож на SAX отсутствием объектной модели в памяти и последовательным продвижением по XML, но в StAX не требуется реализация интерфейсов, и приложение само “командует” StAX-парсеру переход к следующему элементу XML.

Кроме того, в отличие от SAX, данный парсер предлагает API для создания XML-документа.

Работая со StAX можно использовать два типа API

- Iterator API (удобное и простое в использовании)
- Cursor API (быстрое, но низкоуровневое)

Iterator API	Cursor API
XMLEvent XMLEventReader XMLEventWriter	XmlStreamReader XMLStreamWriter

Основными классами StAX Cursor API являются

- javax.xml.stream.**XMLInputFactory**,
- javax.xml.stream.**XMLStreamReader**

и

- javax.xml.stream.**XMLOutputFactory**,
- javax.xml.stream.**XMLStreamWriter**,

которые соответственно используются для чтения и создания XML-документа.

Для чтения XML надо получить ссылку на **XMLStreamReader**:

```
StringReader stringReader = new StringReader(xmlString);
XMLInputFactory inputFactory = XMLInputFactory.newInstance();
XMLStreamReader reader = inputFactory.createXMLStreamReader(stringReader);
```

после чего **XMLStreamReader** можно применять аналогично интерфейсу **Iterator**, используя методы **hasNext()** и **next()**:

- **boolean hasNext()** – показывает, есть ли еще элементы;
- **int next()** – переходит к следующей вершине XML, возвращая ее тип.

Возможные типы вершин:

```
public interface XMLStreamConstants {
	public static final int START_ELEMENT = 1;
	public static final int END_ELEMENT = 2;
	public static final int PROCESSING_INSTRUCTION = 3;
	public static final int CHARACTERS = 4;
	public static final int COMMENT = 5;
	public static final int SPACE = 6;
	public static final int START_DOCUMENT = 7;
	public static final int END_DOCUMENT = 8;
	public static final int ENTITY_REFERENCE = 9;
	public static final int ATTRIBUTE = 10;
	public static final int DTD = 11;
	public static final int CDATA = 12;
	public static final int NAMESPACE = 13;
	public static final int NOTATION_DECLARATION = 14;
	public static final int ENTITY_DECLARATION = 15;
}
```

Далее данные извлекаются применением методов:

- **String getLocalName()** – возвращает название тега;
- **String getAttributeValue(NAMESPACE_URI, ATTRIBUTE_NAME)** – возвращает значение атрибута;
- **String getText()** – возвращает текст тега.

MenuTagName.java

```
package _java._se._13._stax;
public enum MenuTagName {
	NAME, PRICE, DESCRIPTION, CALORIES, FOOD, BREAKFAST_MENU;

	public static MenuTagName getElementTagName(String element) {
		switch (element) {
		case "food":
			return FOOD;
```

```
                case "price":
                        return PRICE;
                case "description":
                        return DESCRIPTION;
                case "calories":
                        return CALORIES;
                case "breakfast-menu":
                        return BREAKFAST_MENU;
                case "name":
                        return NAME;
                default:
                        throw new EnumConstantNotPresentException(MenuTagName.class,
                                        element);
                }
        }
}
```

StAXMenuParser.java

```
package _java._se._13._stax;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamConstants;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

import _java._se._13._sax.Food;

public class StAXMenuParser {

        public static void main(String[] args) throws FileNotFoundException {
                XMLInputFactory inputFactory = XMLInputFactory.newInstance();
                try {
                        InputStream input = new FileInputStream("menu.xml");

                        XMLStreamReader reader =
inputFactory.createXMLStreamReader(input);
                        List<Food> menu = process(reader);

                        for (Food food : menu) {
                                System.out.println(food.getName());
                                System.out.println(food.getCalories());
                        }
                } catch (XMLStreamException e) {
                        e.printStackTrace();
                }

        }

        private static List<Food> process(XMLStreamReader reader)
                        throws XMLStreamException {
                List<Food> menu = new ArrayList<Food>();
                Food food = null;
                MenuTagName elementName = null;
                while (reader.hasNext()) {
                        // определение типа "прочтённого" элемента (тега)
                        int type = reader.next();
                        switch (type) {
```

```
                        case XMLStreamConstants.START_ELEMENT:
                                elementName = MenuTagName.getElementTagName(reader
                                                .getLocalName());
                                switch (elementName) {
                                case FOOD:
                                        food = new Food();
                                        Integer id =
Integer.parseInt(reader.getAttributeValue(
                                                        null, "id"));
                                        food.setId(id);
                                        break;
                                }
                                break;

                        case XMLStreamConstants.CHARACTERS:
                                String text = reader.getText().trim();

                                if (text.isEmpty()) {
                                        break;
                                }
                                switch (elementName) {
                                case NAME:
                                        food.setName(text);
                                        break;
                                case PRICE:
                                        food.setPrice(text);
                                        break;
                                case DESCRIPTION:
                                        food.setDescription(text);
                                        break;
                                case CALORIES:
                                        Integer calories = Integer.parseInt(text);
                                        food.setCalories(calories);
                                        break;
                                }
                                break;

                        case XMLStreamConstants.END_ELEMENT:
                                elementName = MenuTagName.getElementTagName(reader
                                                .getLocalName());
                                switch (elementName) {

                                case FOOD:
                                        menu.add(food);
                                }

                        }

                }
                return menu;
        }
}
```

StAXWriterExample.java

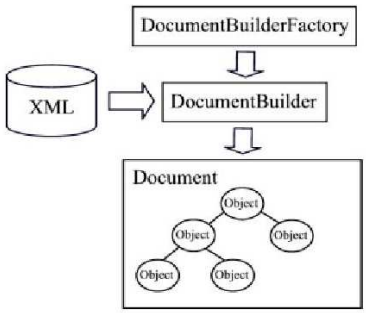
```
package _java._se._13._stax;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.stream.XMLOutputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamWriter;

public class StAXWriterExample {
```

```
        public static void main(String[] args) {
              XMLOutputFactory factory      = XMLOutputFactory.newInstance();

               try {
                   XMLStreamWriter writer = factory.createXMLStreamWriter(
                           new FileWriter("output2.xml"));

                   writer.writeStartDocument();
                   writer.writeStartElement("document");
                   writer.writeStartElement("data");
                   writer.writeAttribute("name", "value");
                   writer.writeCharacters("content");
                   writer.writeEndElement();
                   writer.writeEndElement();
                   writer.writeEndDocument();

                   writer.flush();
                   writer.close();

               } catch (XMLStreamException e) {
                   e.printStackTrace();
               } catch (IOException e) {
                   e.printStackTrace();
               }
        }
}
```

DOM

- DOM фундаментально отличается от SAX.
- DOM представляет собой стандарт, а модель DOM не привязана к Java.
- Существуют частные реализации DOM для JavaScript, Java, CORBA и др.

DOM организован в виде уровней (levels), а не версий. Спецификации DOM можно найти на странице http://www.w3.org/TR/#tr_DOM



DOM Parsing

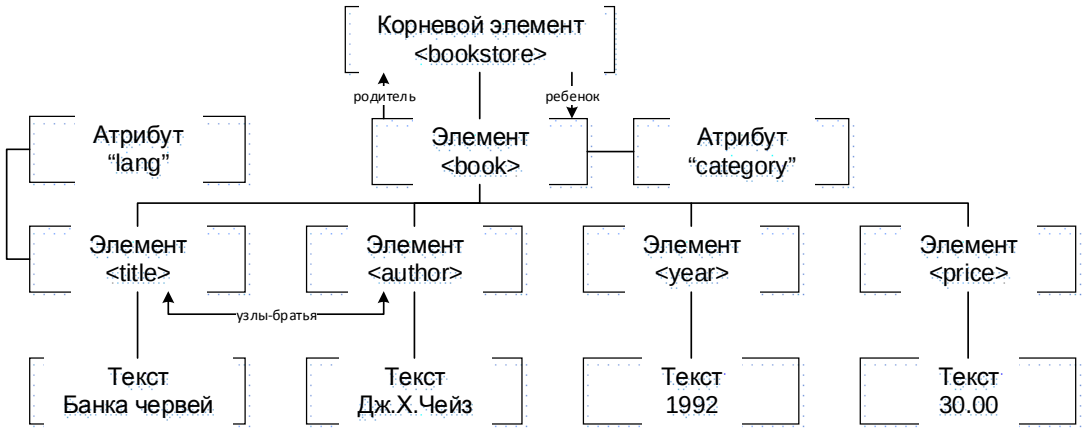
Completed Work

2008-12-22	Element Traversal Specification	Recommendation
2004-04-07	Document Object Model (DOM) Level 3 Load and Save Specification	Recommendation
2004-04-07	Document Object Model (DOM) Level 3 Core Specification	Recommendation
2004-01-27	Document Object Model (DOM) Level 3 Validation Specification	Recommendation
2003-01-09	Document Object Model (DOM) Level 2 HTML Specification	Recommendation
2000-11-13	Document Object Model (DOM) Level 2 Core Specification	Recommendation
2000-11-13	Document Object Model (DOM) Level 2 Events Specification	Recommendation
2000-11-13	Document Object Model (DOM) Level 2 Style Specification	Recommendation
2000-11-13	Document Object Model (DOM) Level 2 Traversal and Range Specification	Recommendation
2000-11-13	Document Object Model (DOM) Level 2 Views Specification	Recommendation
1998-10-01	Document Object Model (DOM) Level 1	Recommendation
2004-02-26	Document Object Model (DOM) Requirements	Group Note
2004-02-26	Document Object Model (DOM) Level 3 Views and Formatting Specification	Group Note
2004-02-26	Document Object Model (DOM) Level 3 XPath Specification	Group Note
2002-07-25	Document Object Model (DOM) Level 3 Abstract Schemas Specification	Group Note

Drafts

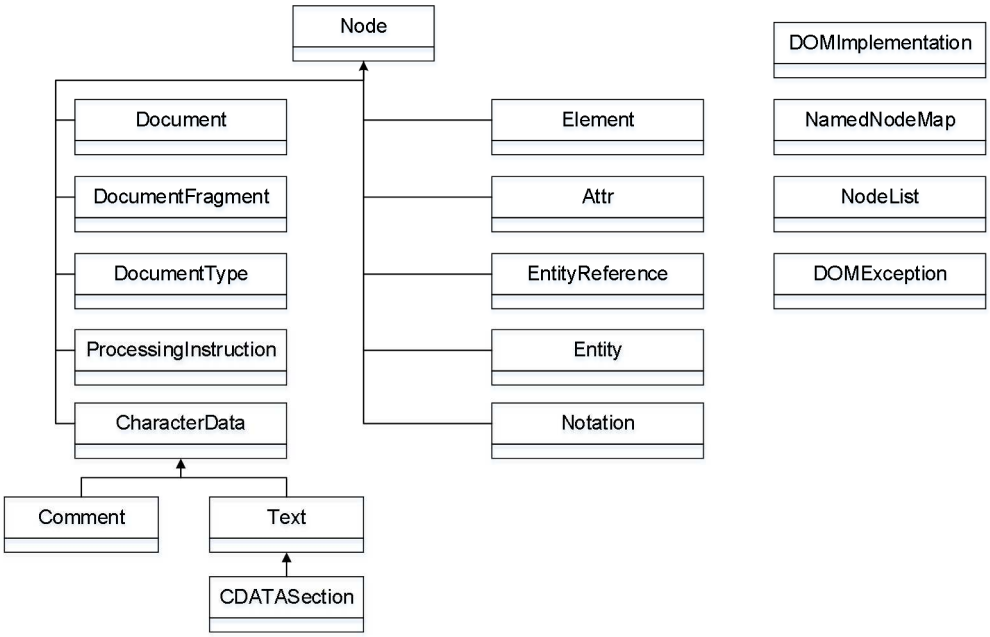
2014-09-25	Document Object Model (DOM) Level 3 Events Specification	Working Draft *Nightly Draft*
2014-07-10	W3C DOM4	Last Call Review ended: 2014-07-31 *Nightly Draft*

Модель DOM представляет XML-документ как древовидную структуру.



Пакет **org.w3c.dom**

UML-модель основных классов и интерфейсов DOM



Существуют различные общепризнанные DOM-анализаторы, которые в настоящий момент можно загрузить с указанных адресов:

Xerces – http://xerces.apache.org/xerces2-j/;

JAXP – входит в JDK.

Существуют также библиотеки, предлагающие свои структуры объектов XML с API для доступа к ним. Наиболее известные:

JDOM – http://www.jdom.org/dist/binary/jdom-1.0.zip.

dom4j – http://www.dom4j.org

org.w3c.dom.Document

Используется для получения информации о документе и изменения его структуры. Это интерфейс представляет собой корневой элемент XML-документа и содержит методы доступа ко всему содержимому документа.

Метод	Назначение
Element getDocumentElement()	возвращает корневой элемент документа

org.w3c.dom.Node

Основным объектом DOM является **Node** – некоторый общий элемент дерева. Большинство DOM-объектов унаследовано именно от **Node**. Для представления элементов, атрибутов, сущностей разработаны свои специализации **Node**.

Интерфейс **Node** определяет ряд методов, которые используются для работы с деревом:

Метод	Назначение
short getNodeType()	возвращает тип объекта (элемент, атрибут, текст, **CDATA** и т.д.);
String getNodeValue()	возвращает значение **Node**
Node getParentNode()	возвращает объект, являющийся родителем текущего узла **Node**

Интерфейс **Node** определяет ряд методов, которые используются для работы с деревом:

Метод	Назначение
NodeList getChildNodes()	возвращает список объектов, являющихся дочерними элементами
Node getFirstChild(), **Node getLastChild()**	возвращает первый и последний дочерние элементы
NamedNodeMap getAttributes()	возвращает список атрибутов данного элемента

У интерфейса **Node** есть несколько важных наследников – **Element**, **Attr**, **Text**. Они используются для работы с конкретными объектами дерева.

org.w3c.dom.Element

Интерфейс предназначен для работы с содержимым элементов XML-документа. Некоторые методы:

Метод	Назначение
String getTagName(String name)	возвращает имя элемента
boolean hasAttribute()	проверяет наличие атрибутов
String getAttribute(String name)	возвращает значение атрибута по его имени
Attr getAttributeNode(String name)	возвращает атрибут по его имени
void setAttribute(String name, String value)	устанавливает значение атрибута, если необходимо, атрибут создается
void removeAttribute(String name)	удаляет атрибут
NodeList getElementsByTagName(String name)	возвращает список дочерних элементов с определенным именем

org.w3c.dom.Attr

Интерфейс служит для работы с атрибутами элемента XML-документа.
Некоторые методы интерфейса **Attr**:

Метод	Назначение
String getName()	возвращает имя атрибута
Element getOwnerElement	возвращает элемент, который содержит этот атрибут
String getValue()	возвращает значение атрибута
void setValue(String value)	устанавливает значение атрибута
boolean isId()	проверяет атрибут на тип ID

org.w3c.dom.Text

Интерфейс **Text** необходим для работы с текстом, содержащимся в элементе.

Метод	Назначение
String getWholeText()	возвращает текст, содержащийся в элементе
void replaceWholeText(String content)	заменяет строкой **content** весь текст элемента

DOMMenuParser.java

```
package _java._se._13._dom;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.xerces.parsers.DOMParser;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import _java._se._13._sax.Food;

public class DOMMenuParser {

	public static void main(String[] args) throws SAXException, IOException{

		//создание DOM-анализатора (Xerces)

		 DOMParser parser = new DOMParser();
		 parser.parse("menu.xml");
		 Document document = parser.getDocument();

		 Element root = document.getDocumentElement();

		 List<Food> menu = new ArrayList<Food>();

			NodeList foodNodes = root.getElementsByTagName("food");
			Food food = null;
			for (int i = 0; i < foodNodes.getLength(); i++) {
				food = new Food();
				Element foodElement = (Element) foodNodes.item(i);

	food.setId(Integer.parseInt(foodElement.getAttribute("id")));
				food.setName(getSingleChild(foodElement,
"name").getTextContent().trim());
				food.setDescription(getSingleChild(foodElement,
"description").getTextContent().trim());
				menu.add(food);
			}

			for (Food f: menu) {
				System.out.println(f.getName() + ", " + f.getId() + ", " +
f.getDescription());
			}

	}

	private static Element getSingleChild(Element element, String childName){
		NodeList nlist = element.getElementsByTagName(childName);
		Element child = (Element) nlist.item(0);
		return child;
	}
}
```

DOMWriterExample.java

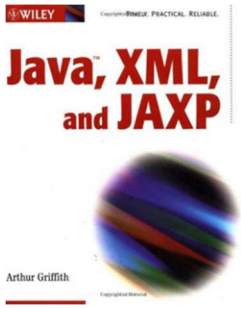
```
package _java._se._13._dom;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
public class DOMWriterExample {

      public static void main(String[] args) throws ParserConfigurationException,
IOException, TransformerException {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
          DocumentBuilder builder = factory.newDocumentBuilder();
          Document document = builder.newDocument();

          Element breakfastMenu = document.createElement("breakfast-menu");
          Element food = document.createElement("food");
          food.setAttribute("id", "234");

          Element name = document.createElement("name");
          name.setTextContent("Waffles");

          food.appendChild(name);
          breakfastMenu.appendChild(food);
          document.appendChild(breakfastMenu);

          TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(document);
        StreamResult result = new StreamResult(new FileWriter("dommenu.xml"));
        transformer.transform(source, result);
      }
}
```

JAXP

JAXP (JavaAPI for XML Processing) – это API

Он не предоставляет новых способов анализа XML и не обеспечивает функциональности для анализа. Он упрощает работу с некоторыми сложными задачами в DOM и SAX.



JAXP предоставляет способ доступа к API SAX и DOM и работы с результатами анализа документа.

Основная цель JAXP – предоставить независимость от производителя при работе с анализаторами.

JAXP имеет все необходимое для создания как SAX-парсеров, так и DOM-парсеров. В дистрибутив JAXP входит анализатор Sun.

JAXP располагается в пакете **javax.xml.parsers**, в состав которого входят четыре класса:

- **DocumentBuilder** — это DOM-парсер, который создает объект класса org.w3c.dom.Document.
- **DocumentBuilderFactory** — класс, который создает DOM-парсеры.
- **SAXParser** — SAX-парсер, который привязывает обработчик SAX-событий к XML-документу, т.е. обрабатывает XML-документ согласно коду, определенному разработчиком.
- **SAXParserFactory** — класс, который создает SAX-парсеры.

Package javax.xml.parsers

Provides classes allowing the processing of XML documents.

See:

Description

Class Summary	
DocumentBuilder	Defines the API to obtain DOM Document instances from an XML document.
DocumentBuilderFactory	Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.
SAXParser	Defines the API that wraps an XMLReader implementation class.
SAXParserFactory	Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents.

Exception Summary	
ParserConfigurationException	Indicates a serious configuration error.

Чтобы обработать XML-документ с помощью парсеров JAXP, как для SAX, так и для DOM необходимо выполнить 4 действия:

1. Получить ссылку на объект одного из Factory-классов (DocumentBuilderFactory или SAXParserFactory).
2. Настроить необходимые параметры и свойства парсера.
3. Создать парсер.
4. Использовать полученный парсер для обработки XML-документа.

JAXP SAX

```
javax.xml.parsers.SAXParserFactory spf =
            SAXParserFactory.newInstance();
spf.setValidating(false);
javax.xml.parsers.SAXParser sp = spf.newSAXParser();
ConcreteSaxHandler handler = new ConcreteSaxHandler();
sp.parse(new File("menu.xml"), handler);
```

JAXP DOM

```
DocumentBuilderFactory dbf =
      DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document document = db.parse("menu.xml");
```

Существуют три метода, позволяющих настроить (включить/выключить) некоторые параметры парсера:

void setNamespaceAware(boolean awareness) — если параметр awareness равен true, то будет создан парсер, который будет учитывать пространства имен, если же awareness равен false, тогда пространства имен учитываться не будут.

void setValidating(boolean validating) — если параметр validating равен true, то парсер, перед тем как приступить к обработке XML-документа, сначала проверит его на соответствие его своему DTD.

void setFeature(String name, boolean value) — этот метод позволяет менять некоторые параметры, определяемые производителями парсера, которым вы пользуетесь (т.е., если вы решили воспользоваться парсером от Oracle, JAXP это позволяет). Парсер должен поддерживать спецификацию SAX2.

Помимо **setValidating** и **setNamespaceAware**, **DocumentBuilderFactory** позволяет также определять следующие параметры:

- **void setCoalescing(boolean value)** — если value равно true, то парсер будет объединять текстовые узлы и CDATA-секции в единые текстовые узлы в DOM-дереве. Иначе CDATA-секции будут вынесены в отдельные узлы.
- **void setExpandEntityReferences(boolean value)** — если установлено (true), то ссылки на сущности будут заменены содержанием этих сущностей (самими сущностями). Если же нет, то эти узлы будут содержать все те же ссылки. Этот метод полезен, если вы не хотите себе головной боли по разыменованию ссылок вручную, если, конечно, они есть.
- **void setIgnoringComments(boolean value)** — если установлено, то все комментарии, содержащиеся в XML-документе, не появятся в результативном DOM-документе. Если нет, тогда DOM-документ будет содержать узлы с комментариями.
- **void setIgnoringElementContentWhitespace(boolean value)** — если установлено, то пробельные символы (символы табуляции, пробелы и пр.), которые располагаются между элементами XML-документа, будут игнорироваться, и они не будут вынесены в узлы результативного DOM-дерева. Если нет, тогда будут созданы дополнительные текстовые узлы, содержащие эти символы.

В DOM нет метода **setFeature**(), как в SAX2, поэтому установка специальных переменных и параметров здесь не предусмотрена.

Замена анализатора

Смена анализатора фактически означает смену конструктора анализатора, поскольку все экземпляры **SAXParser** и **DocumentBuilder** создаются этими конструкторами.

Для замены реализации интерфейса **SAXParserFactory** установите системное свойство Java

javax.xml.parsers.SAXParserFactory.

Если это свойство не определено, возвращается реализация по умолчанию (анализатор, который указал ваш поставщик).

Аналогичный принцип применим и для используемой вами реализации **DocumentBuilderFactory**. В этом случае запрашивается системное свойство

javax.xml.parsers.DocumentBuilderFactory.

TrAX (Trasnformation for API) – API для XML преобразований.

TrAX позволяет использовать таблицы стилей XSL для преобразования XML-документов, а также предоставляет возможность преобразования SAX-событий или DOM-документов в XML-файлы и обратно.

javax.xml.transform
javax.xml.transform.dom
javax.xml.transform.sax
javax.xml.transform.stax
javax.xml.transform.stream

```
TransformerFactory tf = TransformerFactory.newInstance();
Templates template = tf.newTemplates(
                        new StreamSource("newhello.xsl"));
Transformer transformer = template.newTransformer();
transformer.transform(new StreamSource("hello.xml"),
                        new StreamResult("newhello.xml"));
```

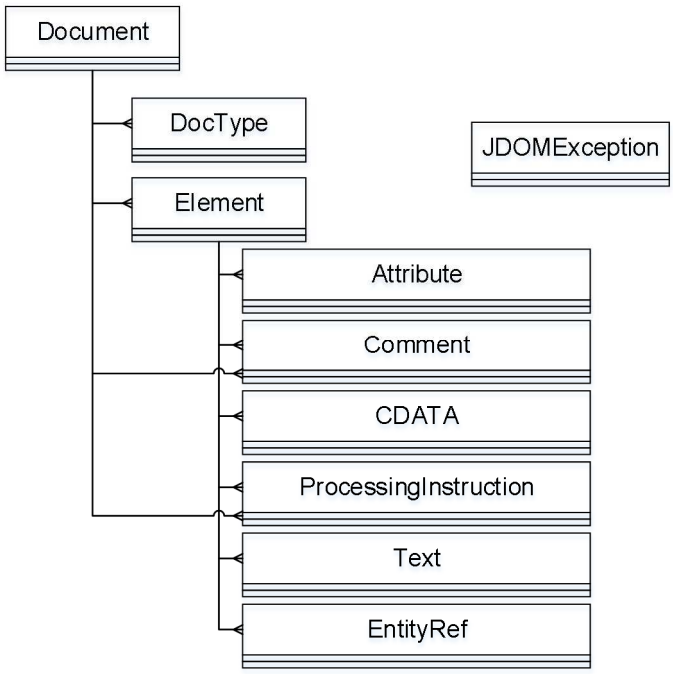
JDOM

JDOM не является анализатором, он был разработан для более удобного, более интуитивного для Java-программист, доступа к объектной модели XML-документа.

JDOM представляет свою модель, отличную от DOM. Для разбора документа JDOM использует либо SAX-, либо DOM-парсеры сторонних производителей.

Реализаций JDOM немного, так как он основан на классах, а не на интерфейсах.

UML-модель основных классов JDOM



org.jdom2.Content

В корне иерархии наследования стоит класс **Content**, от которого унаследованы остальные классы (**Text**, **Element** и др.).

Основные методы класса **Content**:

Метод	Назначение
Document getDocument()	возвращает объект, в котором содержится этот элемент
Element getParentElement()	возвращает родительский элемент

org.jdom2.Document

Базовый объект, в который загружается после разбора XML-документ. Аналогичен **Document** из Xerces.

Метод	Назначение
Element getRootElement()	возвращает корневой элемент.

org.jdom2.Parent

Интерфейс **Parent** реализуют классы **Document** и **Element**. Он содержит методы для работы с дочерними элементами. Интерфейс **Parent** и класс **Content** реализуют ту же функциональность, что и интерфейс **Node** в Xerces.

Некоторые из его методов:

Метод	Назначение
List getContent()	возвращает все дочерние объекты
Content getContent(int index)	возвращает дочерний элемент по его индексу
int getContentSize()	возвращает количество дочерних элементов
Parent getParent()	возвращает родителя этого родителя
int indexOf(Content child)	возвращает индекс дочернего элемента

org.jdom2.Element

Класс **Element** представляет собой элемент XML-документа.

Метод	Назначение
Attribute getAttribute(String name)	возвращает атрибут по его имени
String getAttributeValue(String name)	возвращает значение атрибута по его имени
List getAttributes()	возвращает список всех атрибутов
Element getChild(String name)	возвращает дочерний элемент по имени
List getChildren()	возвращает список всех дочерних элементов
String getChildText(String name)	возвращает текст дочернего элемента
String getName()	возвращает имя элемента
String getText()	возвращает текст, содержащийся в элементе

org.jdom2.Text

Класс **Text** содержит методы для работы с текстом.

Метод	Назначение
String getText()	возвращает значение содержимого в виде строки
String getTextTrim()	возвращает значение содержимого без крайних пробельных символов

org.jdom2.Attribute

Класс **Attribute** представляет собой атрибут элемента XML-документа. В отличие от интерфейса **Attr** из Xerces, у класса **Attribute** расширенная функциональность. Класс **Attribute** имеет методы для возвращения значения определенного типа.

Метод	Назначение
int getAttributeType()	возвращает тип атрибута
тип **get**Тип**Type()**	(**Int**, **Double**, **Boolean**, **Float**, **Long**) возвращает значение определенного типа
String getName()	возвращает имя атрибута
Element getParent()	возвращает родительский элемент

Работа с существующим XML-файлом состоит из следующих этапов:

- Создание экземпляра класса **org.jdom.input.SAXBuilder**, который умеет строить JDOM-дерево из файлов, потоков, URL и т.д.
- Вызов метода **build()** экземпляра SAXBuilder с указанием файла или другого источника.
- Навигация по дереву и манипулирование элементами, если это необходимо.

JDomMenuParser.java

```
import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.input.SAXBuilder;


public class JDomMenuParser {
	public static void main(String[] args) throws JDOMException, IOException {
		SAXBuilder builder = new SAXBuilder();
		Document document = builder.build("menu.xml");

		Element root = document.getRootElement();
		List<Element> menu = root.getChildren();
		Iterator<Element> menuIterator = menu.iterator();
		while (menuIterator.hasNext()){
			Element foodElement = menuIterator.next();
```

```
                System.out.println(foodElement.getChildText("name"));
            }
        }
}
```

JDOM позволяет изменять XML-документ

```
SAXBuilder builder = new SAXBuilder();
Document document = builder.build(filename);
Element root = document.getRootElement();
List c = root.getChildren();
Iterator i = c.iterator();
while (i.hasNext()) {
   Element e = (Element) i.next();
   if (e.getAttributeValue("ш").equals(login)) {
                e.getChild(element).setText(content);
   }
}
XMLOutputter out = new XMLOutputter();
out.output(document, new FileOutputStream(filename));
```

JDOM также позволяет создавать и записывать XML-документы

Для создания документа необходимо создать объект каждого класса (**Element**, **Attribute**, **Document**, **Text** и др.) и присоединить его к объекту, который в дереве XML-документа находится выше.

Element

Для добавления дочерних элементов, текста или атрибутов в элемент XML-документа нужно использовать один из следующих методов:

Метод	Назначение
Element addContent(Content child)	добавляет дочерний элемент
Element addContent(int index, Content child)	добавляет дочерний элемент в определенную позицию
Element addContent(String str)	добавляет текст в содержимое элемента
Element setAttribute(Attribute attribute)	устанавливает значение атрибута
Element setAttribute(String name, String value)	устанавливает атрибут и присваивает ему значение
Element setContent(Content child)	заменяет текущий элемент на элемент, переданный в качестве параметра
Element setContent(int index, Content child)	заменяет дочерний элемент на определенной позиции элементом, переданным как параметр

Element setName(String name)	устанавливает имя элемента
Element setText(String text)	устанавливает текст содержимого элемента

Text

Класс **Text** также имеет методы для добавления текста в элемент XML-документа:

Метод	Назначение
void append(String str)	добавляет текст к уже имеющемуся
void append(Text text)	добавляет текст из другого объекта **Text**, переданного в качестве параметра
Text setText(String str)	устанавливает текст содержимого элемента

Attribute

Методы класса **Attribute** для установки значения, имени и типа атрибута:

Метод	Назначение
Attribute setAttributeType(int type)	устанавливает тип атрибута
Attribute setName(String name)	устанавливает имя атрибута
Attribute setValue(String value)	устанавливает значение атрибута

JomCreateXML.java

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.output.XMLOutputter;

public class JomCreateXML {

	public static void main(String[] args) throws FileNotFoundException,
			IOException {
		Element root = new Element("breakfast-menu");
		Element food = new Element("food");
		food.setAttribute("id", "123");
		Element name = new Element("name");
		name.setText("Waffles");
		food.addContent(name);
		root.addContent(food);
		Document document = new Document(root);
		XMLOutputter outputter = new XMLOutputter();
		outputter.output(document, new FileOutputStream("newmenu.xml"));
	}
}
```

JAXB

Java Architecture for XML Binding (JAXB) – архитектура связывания данных, обеспечивает связь между XML схемами и Java-представлениями, предоставляя возможность использовать данные представленные в виде XML в приложениях Java.

JAXB предоставляет методы для преобразования XML документов в структуры Java и обратно. Кроме этого, есть возможность генерировать XML схемы из Java объектов.

Используя аннотации JAXB конвертирует объекты в/из XML-файл.

- **Marshalling** – конвертирование java-объектов в XML-файл
- **Unmarshalling** – конвертирование XML в java-объект.

Food.java

```
package _java._se._13._jaxb;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Food", propOrder = { "name", "price", "description", "calories" })
public class Food {
	@XmlAttribute(required = true)
	private int id;
	@XmlElement(required = true)
	private String name;
	@XmlElement(required = true)
	private String price;
	@XmlElement(required = true)
	private String description;
	@XmlElement(required = true)
	private int calories;

	public Food(){}
	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPrice() {
		return price;
	}

	public void setPrice(String price) {
```

```
		this.price = price;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public int getCalories() {
		return calories;
	}

	public void setCalories(int calories) {
		this.calories = calories;
	}
}
```

FoodJAXBMarshaller.java

```
package _java._se._13._jaxb;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

public class FoodJAXBMarshaller {

	public static void main(String[] args) throws JAXBException,
FileNotFoundException {
		JAXBContext context = JAXBContext.newInstance(Food.class);
		Marshaller m = context.createMarshaller();

		Food food = new Food();
		food.setId(123);
		food.setName("nnn");
		food.setDescription("ddd");
		food.setCalories(234);
		food.setPrice("333");

		m.marshal(food, new FileOutputStream("stud.xml"));
		m.marshal(food, System.out);
		System.out.println("XML-файл создан");
	}
}
```

FoodJAXBUnMarshaller

```
package _java._se._13._jaxb;
import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

public class FoodJAXBUnMarshaller {

	public static void main(String[] args) throws JAXBException {
		File file = new File("stud.xml");
		JAXBContext jaxbContext = JAXBContext.newInstance(Food.class);

		Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
```

```
			Food food = (Food) jaxbUnmarshaller.unmarshal(file);
			System.out.println(food.getName());
		}
}
```

Валидация

В пакете **javax.xml.validation** для валидации документов используются три класса: **SchemaFactory, Schema** и **Validator**.

Кроме того, этот пакет активно использует интерфейс **javax.xml.transform.Source** для представления документов XML.

XSDValidation.java

```
package _java._se._13._validation;
import java.io.File;
import java.io.IOException;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import org.xml.sax.SAXException;

public class XSDValidation {

	public static void main(String[] args) throws SAXException, IOException {
		// 1. Поиск и создание экземпляра фабрики для языка XML Schema
		SchemaFactory factory = SchemaFactory
				.newInstance("http://www.w3.org/2001/XMLSchema");

		// 2. Компиляция схемы
		// Схема загружается в объект типа java.io.File, но вы также можете
		// использовать
		// классы java.net.URL и javax.xml.transform.Source
		File schemaLocation = new File("src/resources/notes.xsd");
		Schema schema = factory.newSchema(schemaLocation);

		// 3. Создание валидатора для схемы
		Validator validator = schema.newValidator();
		// 4. Разбор проверяемого документа
		Source source = new StreamSource("src/resources/notes.xml");

		// 5. Валидация документа
		try {
			validator.validate(source);
			System.out.println(" is valid.");
		} catch (SAXException ex) {
			System.out.println(" is not valid because ");
			System.out.println(ex.getMessage());
		}

	}
}
```