Лекция 10

Формат сериализованного объекта

Рассмотрим вид сериализованного объекта TestSerial, класс TestSerial имеет вид

```
class parent implements Serializable {
    int parentVersion = 10;}

class contain implements Serializable{
    int containVersion = 11;}

public class SerialTest extends parent implements
                                                Serializable {
    int version = 66;
    contain con = new contain();

    public int getVersion(){
      return version;  }
```

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

   FileOutputStream fos = new FileOutputStream("temp.out");
   ObjectOutputStream oos = new ObjectOutputStream(fos);
   SerialTest st = new SerialTest();
   oos.writeObject(st);
   oos.flush();
   oos.close();
 }
}
```

В файле temp.out сериализованный объект st будет иметь вид

AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65
73 74 05 52 81 5A AC 66 02 F6 02 00 02 49 00 07
76 65 72 73 69 6F 6E 4C 00 03 63 6F 6E 74 00 09
4C 63 6F 6E 74 61 69 6E 3B 78 72 00 06 70 61 72
65 6E 74 0E DB D2 BD 85 EE 63 7A 02 00 01 49 00
0D 70 61 72 65 6E 74 56 65 72 73 69 6F 6E 78 70
00 00 00 0A 00 00 00 42 73 72 00 07 63 6F 6E 74
61 69 6E FC BB E6 0E FB CB 60 C7 02 00 01 49 00
0E 63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E 78
70 00 00 00 0B

Рассмотрим, что представляют собой байты в сериализованном объекте:

AC ED: STREAM_MAGIC. Говорит о том, что используется протокол сериализации.

00 05: STREAM_VERSION. Версия сериализации.

0x73: TC_OBJECT. Обозначение нового объекта.

0x72: TC_CLASSDESC. Обозначение нового класса.

00 0A: Длина имени класса.

53 65 72 69 61 6c 54 65 73 74: SerialTest, имя класса.

05 52 81 5A AC 66 02 F6: SerialVersionUID, идентификатор класса.

0x02: Различные флаги. Этот специфический флаг говорит о том, что объект поддерживает сериализацию.

00 02: Число полей в классе.

алгоритм записывает поле **int version = 66**:

0x49: Код типа поля. 49 это «I», которое закреплено за int.

00 07: Длина имени поля.

76 65 72 73 69 6F 6E: version, имя поля.

Далее алгоритм записывает следующее поле, **contain con = new contain();**

0x74: TC_STRING. Обозначает новую строку.

00 09: Длина строки.

4C 63 6F 6E 74 61 69 6E 3B: Lcontain;, Каноническое JVM обозначаение(т.к. это объект).

0x78: TC_ENDBLOCKDATA, Конец опционального блока данных для объекта.

Затем идет описание класса parent:

0x72: TC_CLASSDESC. Обозначение нового класса

00 06: Длина имени класса.

70 61 72 65 6E 74: parent, имя класса

0E DB D2 BD 85 EE 63 7A: SerialVersionUID, идентификатор класса.

0x02: Различные флаги. Этот флаг обозначает что класс поддерживает сериализацию.

00 01: Число полей в классе.

Далее идет описание полей класса parent, класс имеет одно поле,

int parentVersion = 100

0x49: Код типа поля. 49 обозначает «I», которое закреплено за Int.

00 0D: Длина имени поля.

70 61 72 65 6E 74 56 65 72 73 69 6F 6E: parentVersion, имя поля.

0x78: TC_ENDBLOCKDATA, конец опционального блока данных для объекта.

0x70: **TC_NULL**, обозначает то что больше нет суперклассов, потому что мы достигли верха иерархии классов.

Теперь будут записаны фактические данные ассоциированные с объектом:

00 00 00 0A: 10, значение parentVersion

00 00 00 42: 66, значение version в SerialTest.

Далее записана информация об объекте класса contain.

contain con = new contain();

Для этого делается описание класса contain

0x73: TC_OBJECT, обозначает новый объект.

0x72: TC_CLASSDESC, обозначает новый класс.

00 07: Длина имени класса.

63 6F 6E 74 61 69 6E: contain, имя класса.

FC BB E6 0E FB CB 60 C7: SerialVersionUID, идентификатор этого класса.

0x02: Различные флаги. Этот флаг обозначает что класс поддерживает сериализацию.

00 01: Число полей в классе.

Далее идет описание единственного поля класса conatin, int containVersion = 11;

0x49: Код типа поля. 49 обозначает «I», которое закреплено за Int.

00 0E: Длина имени поля.

63 6F 6E 74 61 69 6E 56 65 72 73 69 6F 6E: containVersion, имя поля.

0x78: TC_ENDBLOCKDATA, конец опционального блока данных для объекта.

Дальше проверяется, имеет ли contain родительский класс. Если имеет, то алгоритм начинает запись этого класса; но в данном случае суперкласса у contain нету, и алгоритм записывает TC_NULL.
0x70: TC_NULL

В конце записываются фактические данные ассоциированные с объектом класса conatin:

00 00 00 0B: 11, значение containVersion.

В будущем сериализация будет заменена форматом XML.

XML

Язык разметки XML (Extensible Markup Language) был разработан W3C.

Главным преимуществом XML является совместимость данных, представленных в этом формате, с различными приложениями.

Язык XML был разработан на базе универсального языка разметки SGML.

Язык HTML, как язык разметки гипертекстовых документов, также произошел от SGML.

Основная идея XML – это текстовое представление с помощью тегов, структурированных в виде дерева данных.

Древовидная структура хорошо описывает бизнес-объекты, конфигурацию, структуры данных и т.п.

Кроме того, представление данных в виде XML удобочитаемо.

DTD

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

DTD определяет, какие теги (элементы) могут использоваться в XML- документе, как эти элементы связаны (например, указывать на то, что элемент **<book>** включает дочерние элементы **<price>** и **<author>**), какие атрибуты имеет тот или иной элемент

Создавать DTD для XML-документа не обязательно, программы-анализаторы будут обрабатывать XML-файл и без DTD.

Но в этом случае автор должен правильно его сформировать.

Для того чтобы сформировать DTD, можно создать либо отдельный файл и описать в нем структуру документа, либо включить DTD-описание непосредственно в документ XML.

В первом случае имеем

<?xml version="1.0" standalone="yes" ?>

<! DOCTYPE journal SYSTEM "book.dtd">

Во втором случае описание элемента помещается в XML-документ:

```
<?xml version="1.0" ?>

...
<! DOCTYPE book [
<!ELEMENT book (price, author)>

...
]>
```

Описание элемента

Элемент в DTD описывается с помощью дескриптора **!ELEMENT**, в котором указывается название элемента и его содержимое.

Например, определим элемент <book> у которого есть дочерние элементы <price> и <author>, тогда получим описание

<!ELEMENT price PCDATA>

<!ELEMENT author PCDATA>

<!ELEMENT book (price, author)>

В данном случае были определены два элемента price и author и описано их содержимое с помощью маркера PCDATA.

Маркер **PCDATA** (parseable character data) указывает анализатору, что элементы могут содержать любую информацию, с которой может работать программа-анализатор.

Кроме маркера PCDATA, есть также маркеры

EMPTY –элемент пуст,

ANY – содержимое документа специально не описывается.

При описании элемента <book>, было указано, что он состоит из дочерних элементов <price> и <author>.

Можно расширить это описание с помощью символов **‘+’, ‘*’, ‘?’,** используемых для указания количества вхождений элементов.

Так, например,

<!ELEMENT book (price, author+, caption?)>

означает, что элемент book содержит один и только один элемент price, несколько (минимум один) элементов author и необязательный элемент caption (символ * указывает на то, что в составе элемента может содержаться любое, в том числе и нулевое количество элементов)

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

Например:

<!ELEMENT book (PCDATA | body)>

в данном случае элемент book может содержать либо дочерний элемент body, либо PCDATA.

Описание атрибутов

Атрибуты элементов описываются с помощью дескриптора !ATTLIST, внутри которого задаются имя атрибута, тип значения, дополнительные параметры:

<!ATTLIST article

id ID #REQUIRED

about CDATA #IMPLIED

type (actual | review | teach) 'actual' >

В данном случае у элемента <article> определяются три атрибута: id, about, type. 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 (**ENTITES**) – содержит имя внешней сущности (несколько имен, разделенных запятыми);

NMTOKEN (**NMTOKENS**) – слово (несколько слов, разделенных пробелами);

Значением атрибута также может быть перечисление.

Набор допустимых значений помещаются в круглые скобки **(значение1| значение2|значение3),** затем указывается значение по умолчанию **‘значение1’** .

Значения по умолчанию могут быть следующими:

#REQUIRED – означает, что значение должно присутствовать в документе;

#IMPLIED – означает, что если значение атрибута не задано, то приложение должно использовать свое собственное значение по умолчанию;

#FIXED – означает, что атрибут может принимать лишь одно значение, то, которое указано в DTD.

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

Определение сущности

Сущность 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.

В XML включено несколько внутренних определений:

&amplt – символ <;

&ampgt – символ >;

&ampamp – символ &;

&ampapos – символ апострофа ‘;

&ampquot – символ двойной кавычки “.

Т.е. объявление помещается между последовательностями символов "<!ENTITY" и ">" и может быть представлено в одном из перечисленных ниже форматов.

Внутренний примитив. Имени примитива ставится в соответствие значение, используемое в XML-документе.

Если в составе документа встречается выражение **&имя_примитива**, оно заменяется значением, связанным с именем.

<! ENTITY имя_примитива "значение_примитива">

Внешний примитив XML. Имени примитива ставится в соответствие XML документ, URL которого указывается в при определении примитива.

Если в составе документа встречается выражение **&имя_примитива;**, оно заменяется содержимым XML-документа.

В процессе чтения документа производится его разбор

<!ENTITY имя_примитива SYSTEM " URL">

Внешний двоичный примитив. Имени примитива ставится в соответствие набор двоичных данных с заданным URL.

Если при обработке документа встречается выражение

&имя примитива;, оно заменяется данными, тип которых указан в объявлении примитива.

Так, если в объявлении задан тип GIF87A, это означает, что URL указывает на двоичный файл в формате GIF.

В процессе чтения разбор содержимого внешнего двоичного примитива не производится и он может быть использован лишь в атрибуте элемента.

<!ENTITY имя_примитива SYSTEM "URL" NDATA тип_данных>

Пример:

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

Примитив параметра. Имя примитива связывается со значением, которое может быть использовано лишь в составе DTD (но не в XML-документе).

Перед именем такого примитива указывается символ "%", а ссылка на значение примитива задается в DTD в виде %имя примитива DTD).

<!ENTITY %имя примитива DTD

"значение_примитива_DTD">

Пусть существует XML-документ, содержащий данные адресной книги:

```
<?xml version="1.0"?>
  <!DOCTYPE notepad SYSTEM "notepad.dtd">
<notepad>
 <note login="rom">
    <name>Valera</name>
    <tel> 217819</tel>
    <url> http://www.aaa.com </url>
   <address>
    <street> Main Str., 35 </street>
    <city> Kiev
    <country> UKR </country>
   </address>
 </note>
```

```
<note login="goch">
  <name> Igor </name>
  <tel> 430797 </tel>
   <url> http://http://www.a.com </url>
  <address>
    <street> Deep Forest, 7</street>
     <city> Polock </city>
     <country> VCL </country>
   </address>
 </note>
</notepad>
```

Тогда DTD файл будет иметь вид:

```
<?xml version="1.0" encoding="UTF-8"?>
  <!ELEMENT notepad (note+)>
  <!ELEMENT note (name,tel,url,address)>
  <!ELEMENT address (street,city,country)>
  <!ATTLIST note login ID #REQUIRED>
  <!ELEMENT name PCDATA>
  <!ELEMENT tel PCDATA>
  <!ELEMENT street PCDATA>
  <!ELEMENT city PCDATA>
  <!ELEMENT country PCDATA>
  <!ELEMENT url PCDATA>
```

XLink

Гипертекстовые средства XML базируются на понятии ресурса.

Ресурсами считаются адресуемые фрагменты данных (например, файлы, изображения и приложения).

Связи, или ссылки, отражают отношения между ресурсами.

Локальным ресурсом называется ресурс, который включается в состав связующего XML-элемента.

Удаленный ресурс — это фрагмент данных, на который указывает XML-ссылка.

Связующими элементами называются XML-элементы, в состав которых входят ссылки.

Для реализации ссылки используется атрибут с именем **xlink:type**.

Стандартные значения атрибута **xlink:type**, которые определены спецификацией XLink, перечислены ниже.

simple. Создает простой связующий элемент, определяющий связь между локальным и удаленным ресурсами.

extended. Создает расширенный связующий элемент, определяющий связь между несколькими ресурсами.

locator. Элемент, определяющий адрес удаленного ресурса.

В составе расширенной ссылки типа extended должен быть задан хотя бы один элемент locator.

arc. Определяет правила перехода между ссылками, содержащимися в расширенном связующем элементе (типа extended).

title. Задает символьное описание расширенной ссылки.

resource. Определяет локальный ресурс для внешней ссылки.

Если внешняя ссылка использует элемент типа resource, она называется внутренней (inline).

Простые ссылки

Простая ссылка устанавливает связь между локальным и внешним ресурсами.

Формат простой ссылки имеет вид:

<имя_элемента xlink:type="simple" xlink:href=“URL”>

.....

</имя_элемента>

или

<имя элемента xlink:type="simple" xlink:href="URL"/>

Для создание простой ссылки используется значение **simple** атрибута **xlink:type**.

В качестве значения атрибута **xlink:href** указывается URL ресурса.

Например:

<homepage xlink:type="simple"

xlink:href=“http://www.google.com">

Google Home

</homepage>

Простые ссылки во многом напоминают ссылки, применяемые в HTML-документах.

Связующий элемент, построенный посредством простой ссылки, может отображаться в программах просмотра (например, в Web-броузерах), а путем активизации ссылки (например, по щелчку мыши) можно обратиться к удаленному ресурсу.

Атрибуты, определяющие поведение ссылок

Кроме атрибутов, используемых для связывания ресурсов, существуют атрибуты, позволяющие определить дополнительные характеристики связующих элементов:

xlink:title- атрибут задает символьное имя удаленного ресурса;

xlink:role- атрибут определяющий назначение удаленного ресурса.

Значение атрибута **xlink:role** используется при обработке документа.

В составе связующего элемента могут также присутствовать атрибуты, задающие поведение ссылки.

xlink:show- задает порядок отображения ресурса, на который указывает ссылка.

Значение **xlink: show="new"** сообщает о том, что удаленный ресурс должен отображаться в новом окне.

Если задано значение **xlink:show="replace",** удаленный ресурс замещает содержимое текущего окна.

Значение **xlink:show="embed"** говорит о том, что ресурс, на который указывает ссылка, должен встраиваться в текущий документ.

xlink:actuate- задает условия активации ссылки.

Например, если в документе задано значение **xlink:actuate="onLoad",** это означает, что приложение должно загрузить удаленный ресурс сразу после разбора ссылки.

Значение **link:actuate="onRequest"**

указывает на то, что ресурс должен загружаться после того, как пользователь активирует ссылку (например, щелкнет на ней мышью).

Расширенные ссылки

Расширенные ссылки могут использоваться для создания множественных связей.

Если одна из связей указывает на локальный элемент, ссылка называется внутренней.

Если все связи указывают на удаленные ресурсы, ссылка называется внешней.

Для идентификации расширенной ссылки используется значение **extended** атрибута **xlink:type**.

Локальный ресурс указывается в составе расширенной ссылки как элемент с атрибутом **xlink:type="resource".**

Удаленный ресурс, находящийся за пределами расширенной ссылки (например, в составе другого XML-документа), представляется посредством подчиненного элемента с атрибутом **xlink:type="locator".**

Для идентификации ресурса в элементе такого типа используется URL, который задается в качестве значения атрибута **xlink:href**.

Рассмотрим несколько примеров.

Предположим, что необходимо выразить на XML отношение между художником и окружающей его обстановкой.

Это подразумевает создание связей между этим творческим работником и его наследием, а также задание связи к описанию исторических событий, имевших место на протяжении его жизни.

Пусть данные о художнике записаны в следующем файле:

```
<?xml version="1.0"?>
<artistinfo>
   <surname> Modigliani </surname>
   <name> Amadeo </name>
   <born> July 12, 1884 </born>
   <died> January 24, 1920 </died>
   <biography>
        <p> In 1906, Modigliani settled in Paris,
                                                where ... </p>
   </biography>
</artistinfo>
```

Помимо этого, в отдельные файлы включаются описания периодов, на которые можно условно разбить его творчество:

```
<?xml version="1.0"?>
<period>
   <city> Paris </city>
  <country> France <country>
  <timeframe begin="1900" end="1920"/>
  <title> Paris in the early 20th century (up to the
                                          twenties) </title>
  <end> Amadeo </end>
  <description>
       <p> During this period, Russian, Italian, ...</p>
   </description>
</period>
```

Рассмотрим решение задачи- создание связей между этим творческим работником и его наследием с помощью **XLink.**

Как уже было сказано, в XLink используются два типа связующих элементов (linking elements): **simple** (простой) - подобный "a" и "img" в HTML

и **extended** (расширенный).

Т.е.

```
<environment xlink:type="extended">
 <!-- Это расширенная связь -->
 <!-- Здесь должны быть включены
                задействованные ресурсы -->
</environment>
```

После объявления расширенной связи, необходимо указать задействованные ресурсы.

Поскольку информация о художнике и его жизни хранится вне документа описывающего художника (и, следовательно, ею невозможно управлять), чтобы ссылаться на нее, необходимо использовать элементы XLink, атрибуты которых имеют значение **locator**.

Т.е. имеем

```
<environment xmlns:xlink="http://www.w3.org/1999/xlink"
                                                    xlink:type="extended">

   <!-- Ресурсы, задействованные в нашей связи, - художник -->
   <!-- он сам, его наследие и периоды творчества -->

<artist xlink:type="locator" xlink:label="artist" xlink:href="modigliani.xml"/>

<influence xlink:type="locator" xlink:label="inspiration"

    xlink:href="cezanne.xml"/>

<influence xlink:type="locator" xlink:label="inspiration" xlink:href="lautrec.xml"/>

<influence xlink:type="locator" xlink:label="inspiration" xlink:href="rouault.xml"/>

 <history xlink:type="locator" xlink:label="period" xlink:href="paris.xml"/>

 <history xlink:type="locator" xlink:label="period" xlink:href="kisling.xml"/>

</environment>
```

Уточним данный файл указав отношение между ресурсами:

```
<environment xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">

<!-- художник привязан к своему наследию и периодам творчества -->

  <artist xlink:type="locator" xlink:role="artist" xlink:href="modigliani.xml"/>

  <influence xlink:type="locator" xlink:label="inspiration“  xlink:href="cezanne.xml"/>

  <influence xlink:type="locator" xlink:label="inspiration“   xlink:href="lautrec.xml"/>

  <influence xlink:type="locator" xlink:label="inspiration“   xlink:href="rouault.xml"/>

  <history xlink:type="locator" xlink:label="period" xlink:href="paris.xml"/>
  <history xlink:type="locator" xlink:label="period" xlink:href="kisling.xml"/>

  <bind xlink:type="arc" xlink:from="artist" xlink:to="inspiration"/>
  <bind xlink:type="arc" xlink:from="artist" xlink:to="period"/>

</environment>
```

Рассмотрим пример документа, содержащего три ссылки на удаленные ресурсы, описывающих разновидности продукта.

Значения атрибута **xlink:role** могут быть использованы для выбора ресурса наиболее соответствующего интересам заказчика:

```
<sweaters xlink:type="extended"
                                    xlink:title="BeeShirts.com Sweaters>
  <sweaterAd xlink:type="resource“  xlink:role="sweaters">
  BeeShirts has sweater styles to suit you
</sweaterAd>
<default xlink:type=“locator“
  xlink:href=“http://www.beeshirts.com/sweaters/standart/"
                                                xlink:role="standard"/>
```

```
<option xlink:type="locator“
                xlink:href=“http://www.beeshirts.com/sweaters/calsic/“
                  xlink:title="Classic Sweater Style“ xlink:role="classic"/>

<option xlink:type="locator"
              xlink:href=“http://www.beeshirts.com/sweaters/sports/“
                  xlink:title="Sporty Sweater Style“ xlink:role="sports"/>
</sweaters>
```

Правила перехода будут иметь вид:

```
<sweaters xlink:type=“extended”
                xlink:title=“BeeShirts.com Sweaters”>
.....
<selection xlink:type="arc"
   xlink:from="sweaters" xlink:to="classic"
   xlink:show="replace"
   xlink:actuate="onRequest"/>
</sweaters>
```

Рассмотрим пример простых связей.

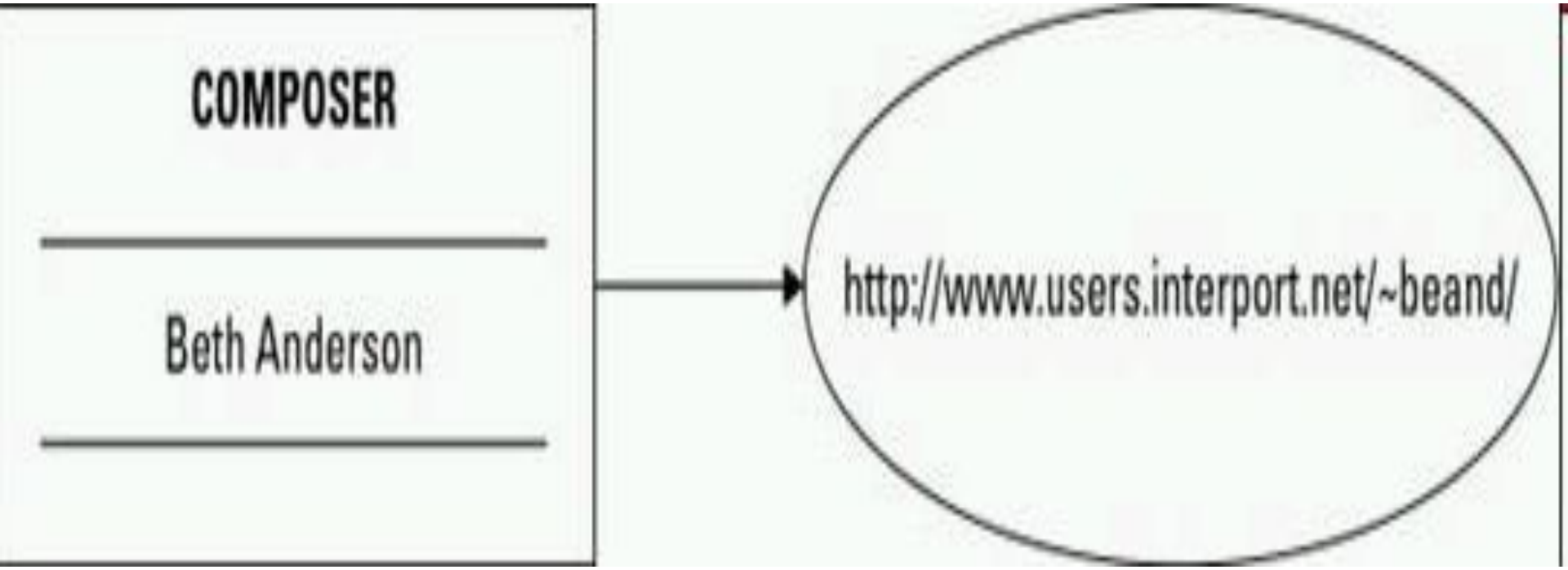
```
<COMPOSER xmlns:xlink="http://www.w3.org/1999/xlink"
          xlink:type="simple"
          xlink:href="http://www.users.interport.net/~beand/">
   Beth Anderson
</COMPOSER>

<FOOTNOTE xmlns:xlink="http://www.w3.org/1999/xlink"
                  xlink:type="simple" xlink:href="footnote7.xml">
  7
</FOOTNOTE>

<IMAGE
  xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple“
  xlink:href="logo.gif"
  xlink:actuate="onLoad"
  xlink:show="embed"/>
```

Например, в первом элементе COMPOSER атрибут xlink:href определяет адресат связи.

Значение атрибута - абсолютный URL http://www.users.interport.net/~beand/.

Этот связующий элемент описывает соединение элемента COMPOSER текущего документа, содержание которого "Beth Anderson", с удаленным документом в http://www.users.interport.net/~beand/.

На рисунке это можно изобразить следующим образом:


COMPOSER
Beth Anderson
http://www.users.interport.net/~beand/

Рассмотрим примеры использования атрибута xlink:show.

```
<COMPOSER  xlink:type="simple“ xlink:show="replace“
      xlink:href="http://www.users.interport.net/~beand/">
   Beth Anderson
</COMPOSER>
```

Если значение xlink:show равно replace, то при активизации связи (как правило, посредством щелчка мышкой по этой связи, например, в браузерах) адресат связи заменяет текущий документ в том же самом окне.

<WEBSITE
xlink:type="simple"
xlink:show="new"
xlink:href="http://www.quackwatch.com/">
Check this out, but don't leave our site completely!
</WEBSITE>

Если значение xlink:show равно new, то активизация связи вызывает открытие нового окна, в котором отображается адресуемый ресурс.

```
<PHOTO
  xlink:type="simple"
 xlink:href="images/nypride.jpg"
 xlink:show="embed"
 ALT="Marchers on 5th Avenue, June 2000"/>
```

Если значение xlink:show равно embed, то при активизации связи адресуемый ресурс вставляется в существующий документ.

Что именно это означает - зависит от приложения.

XSD

Формат xsd идет на смену формату dtd.

Рассмотрим пример. Пусть имеется файл “заказ на покупку” po.xml:

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20“
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=“po.xsd">
  <shipTo country="US">
    <name> Alice Smith </name>
    <street> 123 Maple Street </street>
    <city> Mill Valley </city>
    <state> CA </state>
    <zip> 90952 </zip>
  </shipTo>
  <billTo country="US">
    <name> Robert Smith </name>
    <street> 8 Oak Avenue </street>
    <city> Old Town </city>
    <state> PA </state>
    <zip> 95819 </zip>
  </billTo>
  <comment> Hurry, my lawn is going wild </comment>
```

```
 <items>
  <item partNum="872-AA">
   <productName> Lawnmower </productName>
   <quantity> 1 </quantity>
   <USPrice> 148.95 </USPrice>
   <comment> Confirm this is electric </comment>
  </item>
  <item partNum="926-AA">
   <productName> Baby Monitor </productName>
   <quantity> 1 </quantity>
   <USPrice> 39.98 </USPrice>
   <shipDate> 1999-05-21 </shipDate>
  </item>
 </items>
</purchaseOrder>
```

Пусть схема документа "Заказ на покупку" содержится в файле po.xsd.

```
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
   <xsd:documentation xml:lang="en"> Purchase order
   schema for Example.com. Copyright 2000
   Example.com. All rights reserved.
   </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="purchaseOrder"
                              type="PurchaseOrderType"/>
  <xsd:element name="comment" type="xsd:string"/>
  <xsd:complexType name="PurchaseOrderType">
```

```
<xsd:sequence>
  <xsd:element name="shipTo" type="USAddress"/>
  <xsd:element name="billTo"  type="USAddress"/>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items" type="Items"/>
</xsd:sequence>
<xsd:attribute name="orderDate"  type="xsd:date"/>
</xsd:complexType>
```

```
<xsd:complexType name="USAddress">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city" type="xsd:string"/>
   <xsd:element name="state" type="xsd:string"/>
   <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country"
                 type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

```
<xsd:complexType name="Items">
 <xsd:sequence>
  <xsd:element name="item" minOccurs="0"
                                    maxOccurs="unbounded">
<xsd:complexType>
 <xsd:sequence>
 <xsd:element name="productName" type="xsd:string"/>
 <xsd:element name="quantity">
  <xsd:simpleType>
   <xsd:restriction base="xsd:positiveInteger">
     <xsd:maxExclusive value="100"/>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:element>
```

```
    <xsd:element name="USPrice“ type="xsd:decimal"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="shipDate" type="xsd:date"
                                          minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="partNum" type="SKU"
                                          use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

```
 <xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
  <xsd:pattern value="\d{3}-[A-Z]{2}"/>
 </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

Схема заказа на покупку состоит из элемента schema и множества подэлементов, среди которых наиболее часто упоминаются element, complexType и simpleType.

Элементы схемы определяют порядок следования элементов и их содержание в документах типа "Заказ на покупку".

Каждый из элементов в схеме имеет префикс xsd:.

Этот префикс связан с именным пространством XML-схемы через объявление xmlns:xsd=http://www.w3.org/2001/XMLSchema, которое задано в элементе schema.

Префикс xsd: используется в соответствии с соглашением об использовании этого именного пространства для обозначения элементов XML-схемы, хотя можно использовать любой префикс.

Тот же самый префикс, и следовательно, та же самая ассоциация с именным пространством, используется и в названиях встроенных простых типов.

Например, **xsd:string**.

Определение комплексных типов, объявление элементов и атрибутов

Новые комплексные типы определяются с помощью оператора **complexType**.

Такие определения обычно содержат набор из объявлений элементов, ссылок на элементы, и объявлений атрибутов.

Объявления не задают самостоятельно типы. Они создают ассоциации между именем элемента и ограничениями, которые управляют появлением этого имени в документах, соответствующих данной схеме.

Элементы объявляются, с помощью оператора **element**.

Атрибуты объявляются, с помощью оператора **attribute**

Рассмотрим определение комплексного типа USAddress:

```
<xsd:complexType name="USAddress" >
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city" type="xsd:string"/>
   <xsd:element name="state" type="xsd:string"/>
   <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country"
                 type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

В результате этого определения любой элемент типа USAddress, появляющийся в документе (например, элемент shipTo в файле po.xml), должен состоять из пяти элементов и одного атрибута.

Имена этих пяти элементов (name, street, city, state и zip) объявляются с помощью атрибута name оператора element, **причем элементы должны появиться в той же самой последовательности, в которой они объявлены.**

Определение USAddress содержит объявления, включающие только простые типы: string, decimal и NMTOKEN.

А определение PurchaseOrderType содержит объявления элементов, имеющих комплексные типы. Например, USAddress.

Оба вида объявлений (простые и комплексные) используют тот же самый атрибут type.

Определение типа PurchaseOrderType имеет вид:

```
<xsd:complexType name="PurchaseOrderType">
 <xsd:sequence>
  <xsd:element name="shipTo"  type="USAddress"/>
  <xsd:element name="billTo"   type="USAddress"/>
  <xsd:element ref="comment" minOccurs="0"/>
  <xsd:element name="items" type="Items"/>
 </xsd:sequence>
 <xsd:attribute name="orderDate"  type="xsd:date"/>
</xsd:complexType>
```

В определении PurchaseOrderType, объявления элементов shipTo и billTo, связывают различные имена элементов с одним и тем же комплексным типом, а именно с USAddress.

Элементы shipTo и billTo могут иметь атрибут country, который был объявлен как часть определения USAddress.

Определение PurchaseOrderType содержит объявление атрибута orderDate, который, подобно объявлению атрибута country, задается с помощью простого типа.

Фактически, все объявления атрибутов должны выполняться с помощью простых типов, потому что, в отличие от элементов, атрибуты не могут содержать другие элементы или другие атрибуты.

Иногда предпочтительно использовать ссылку на существующий тип элемента, а не объявлять новый, например:

< xsd:element ref="comment" minOccurs="0"/>

В этом объявлении приводится ссылка на существующий элемент comment, который объявлен где-то в другом месте схемы заказа на закупку.

Значение атрибута **ref** должно рассматриваться, как ссылка на глобальный элемент, который был объявлен в элементе schema, а не как часть определения комплексного типа.

Вследствие этого элемент comment может появиться в документе внутри элемента PurchaseOrderType, причем его содержание должно быть совместимо с типом string.

Ограничение вхождений

Значение параметра minOccurs равное 0 у элемента comment говорит о том, что он не обязательно будет присутствовать в составе элемента PurchaseOrderType.

Вообще, элемент является обязательным, если значение **minOccurs** больше или равно 1.

Максимальное число появлений элемента определяется значением, задаваемым параметром **maxOccurs**.

Это значение может быть положительным целым числом, или термом unbounded, что означает отсутствие ограничения максимального числа появлений.

Значение по умолчанию для minOccurs и для maxOccurs равно 1.

Атрибуты, в отличие от элементов, могут появиться только однажды или ни разу.

В частности атрибуты могут быть объявлены с параметром **use**.

В зависимости от значения этого параметра атрибут

обязателен (use="required"),

необязателен (use="optional"),

запрещен (use="prohibited").

Значения по умолчанию и атрибутов и элементов могут быть объявлены с использованием параметра **default**, хотя этот параметр в том или ином случае работает по разному.

Атрибут со значением, определенным по умолчанию, может появляться или не появляться в xml документе.

Если атрибут не появляется в документе, то обработчик схемы обеспечивает, атрибут со значением равным значению default.

Значение по умолчанию для элементов обрабатывается немного по-другому.

Если элемент появляется в документе, но не содержит какого либо значения, то в качестве его значения подставляется значение по умолчанию.

Атрибут **fixed** используется в объявлениях и атрибутов и элементов.

Он используется, чтобы указать, что атрибут или элемент могут принимать фиксированные значения.

Простые типы

Язык XML-схемы имеет довольно обширный набор простых типов.

Новые простые типы можно определить, получая их от существующих простых типов (встроенных или ранее определенных).

В частности можно получить новый простой тип, ограничивая существующий простой тип.

Другими словами, для нового типа можно установить собственный диапазон значений как подмножество диапазона значений существующего типа.

Рассмотрим пример, создадим новый простой тип myInteger:

```
<xsd:simpleType name="myInteger">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="10000"/>
    <xsd:maxInclusive value="99999"/>
  </xsd:restriction>
</xsd:simpleType>
```

Чтобы определить тип myInteger, ограничивается диапазон базового типа integer, используя два фасета, названные minInclusive и maxInclusive.

Возможно использование и других комбинаций встроенных простых типов и фасетов.

Рассмотрим более сложный пример определения простого типа.

Тип по имени SKU получен из простого типа string. Значения SKU ограничивается путем использования фасета pattern, который содержит регулярное выражение, определяющее допустимый формат строки "\d{3}-[A-Z]{2}".

```
<xsd:simpleType name="SKU">
 <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-[A-Z]{2}"/>
 </xsd:restriction>
</xsd:simpleType>
```

Язык XML-схем определяет пятнадцать фасетов.

Среди них особенно полезен фасет **enumeration**. Его можно использовать для ограничения значения почти каждого простого типа, кроме boolean.

Фасет **enumeration** ограничивает простой тип набором явных значений.

Рассмотрим пример:

```
<xsd:simpleType name="USState">
 <xsd:restriction base="xsd:string">
   <xsd:enumeration value="AK"/>
   <xsd:enumeration value="AL"/>
   <xsd:enumeration value="AR"/>
   <!-- and so on ... -->
 </xsd:restriction>
</xsd:simpleType>
```

Тип List

В дополнение к так называемым атомарным типам, которые составляют большинство, XML-схема имеет понятие списка.

Списочные типы состоят из последовательностей атомарных типов, и, следовательно, допустимыми значениями могут быть только "атомы" из этой последовательности.

Например, списочный тип NMTOKENS состоит из значений типа NMTOKEN, разделенных пробелами.

В дополнение к встроенным списочным типам можно создать новые списочные типы из существующих атомарных типов.

Невозможно создать списочные типы из существующих списочных типов или из комплексных типов.

Рассмотрим пример создания списочного типа:

```
<xsd:simpleType name="listOfMyIntType">
    <xsd:list itemType="myInteger"/>
</xsd:simpleType>
```

Элемент в документе, содержимое которого соответствует типу listOfMyIntType, может выглядеть следующим образом:

<myTag>20003 15037 95977 95945</myTag>

Для создания списочного типа могут быть применены следующие фасеты: **length, minLength, maxLength,** и **enumeration**.

Например, чтобы определить список точно из шести штатов США (SixUSStates), необходимо сначала определить новый списочный тип (полученный из типа USState) с именем USStateList, а затем создать тип SixUSStates, ограничивая USStateList только шестью элементами.

Списочный тип SixUSStates:

```
<xsd:simpleType name="USStateList">
  <xsd:list itemType="USState"/>
</xsd:simpleType>

<xsd:simpleType name="SixUSStates">
 <xsd:restriction base="USStateList">
  <xsd:length value="6"/>
 </xsd:restriction>
</xsd:simpleType>
```

Элементы, тип которых - SixUSStates, должны содержать шесть элементов, и каждый из этих шести элементов должен быть одним из атомарных значений перечислимого типа USState, например:

```
<sixStates>PA NY CA NY LA AK</sixStates>
```

Тип Union

Атомарные типы и списочные типы дают возможность элементу или атрибуту принимать значение (одно или более) экземпляра одного атомарного типа.

Тип **Union** дает возможность элементу или атрибуту принимать значение (одно или более) одного типа, образованного путем объединения множества атомарных и списочных типов.

Например, создадим union-тип для идентификации штатов США как односимвольного сокращения названия или списка числовых кодов.

```
<xsd:simpleType name="zipUnion">
  <xsd:union
        memberTypes="USState listOfMyIntType"/>
</xsd:simpleType>
```

Атрибут memberTypes оператора union задает список всех типов в объединении.

Предположим, что был объявлен элемент с названием zips типа zipUnion, тогда он может принимать следующие значения:

```
<zips>CA</zips>
<zips>95630 95977 95945</zips>
<zips>AK</zips>
```

Определение анонимных типов

При создании схем применяется два стиля.

Схемы могут создаваться путем определения поименованных типов (например, PurchaseOrderType) с последующим объявлением элементов этого типа (например, purchaseOrder).

При этом объявленные элементы ссылаются на поименованный тип с помощью конструкции type= .

Этот стиль является достаточно простым, но может стать неуправляемым, особенно если определяется много типов, на которые ссылаются только один раз, и которые содержат немного ограничений.

В этих случаях, тип может быть более кратко определен как анонимный.

Анонимный тип нет необходимости именовать и, следовательно, задавать на него ссылки.

Определение типа **Items** в po.xsd содержит два объявления **item** и **quantity**, использующие анонимный тип.

Комплексные типы из простых типов

Рассмотрим вопрос как задать определение комплексного типа, который основан на простом типе, например, **decimal** ?

Очевидно, необходимо получить новый комплексный тип из простого типа decimal.

Рассмотрим пример:

```
<xsd:element name="internationalPrice">
 <xsd:complexType>
  <xsd:simpleContent>
   <xsd:extension base="xsd:decimal">
    <xsd:attribute name="currency" type="xsd:string"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>
</xsd:element>
```

Для того чтобы начать описание нового анонимного типа, используется элемент complexType.

Чтобы указать, что новый тип содержит только символьные данные и не содержит подэлементов, используется элемент simpleContent.

Наконец, получаем новый тип, расширяя простой тип decimal.

Расширение типа decimal заключается в добавлении (путем использования стандартного объявления) атрибута currency.

Смешанное содержимое

Рассмотрим вариант размещения символьных данных в любом элементе. Пример:

```
<letterBody>
 <salutation> Dear Mr.
    <name> Robert Smith</name> .
</salutation> Your order of
   <quantity> 1 </quantity>
   <productName>
      Baby Monitor
  </productName>
   shipped from our warehouse on
  <shipDate> 1999-05-21</shipDate>. ....
</letterBody>
```

Текст появляется между элементами salutation, quantity, productName и shipDate, которые являются дочерними элементами letterBody.

Рассмотрим схему для документа letterBody.

```
<xsd:element name="letterBody">
  <xsd:complexType mixed="true">
   <xsd:sequence>
    <xsd:element name="salutation">
    <xsd:complexType mixed="true">
     <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="quantity" type="xsd:positiveInteger"/>
  <xsd:element name="productName" type="xsd:string"/>
  <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
 <!-- etc. -->
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Элементы, появляющиеся в письме клиенту объявлены, и их типы определены, с помощью операторов element и complexType.

Чтобы разрешить символьным данным появиться между дочерними элементами letterBody, атрибут **mixed** в операторе определения типа равен **true**.

Пустое содержимое

Предположим, что элемент internationalPrice будет задавать наименование валюты и цену как значения атрибутов, а не как значение атрибута и содержимого элемента.

Например:

<internationalPrice currency="EUR" value="423.46"/>

Такой элемент вообще не имеет никакого содержания. Чтобы определить тип, содержание которого пусто, необходимо определить тип, который позволяет включать в его состав только подэлементы, но при этом не объявлять никаких элементов.

xsd схема для такого элемента имеет вид:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
   <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
     <xsd:attribute name="currency"  type="xsd:string"/>
     <xsd:attribute name="value"  type="xsd:decimal"/>
    </xsd:restriction>
   </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

В этом примере, определяется анонимный тип с помощью оператора **complexContent**, то есть предполагается, что он будет содержать только элементы.

Оператор **complexContent** говорит о том, что модель комплексного типа будет ограничиваться или расширяться, а элемент **restriction** с параметром **anyType** объявляет два атрибута, но не задает никакого содержания элемента.

Предыдущий синтаксис для объявления пустого элемента является относительно подробным.

Элемент internationalPrice можно объявить короче.

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
    <xsd:attribute name="currency"  type="xsd:string"/>
    <xsd:attribute name="value"  type="xsd:decimal"/>
  </xsd:complexType>
</xsd:element>
```

Этот компактный синтаксис работает потому, что комплексный тип, определенный как simpleContent или complexContent интерпретируется как упрощенное описание комплексного типа, который по умолчанию ограничивается параметром **anyType**.

anyType

anyType представляет абстракцию, которая является базовым типом прародителем всех простых и комплексных типов.

Тип anyType не ограничивает как-либо свое содержимое.

Тип anyType используется подобно другим типам, например:

<xsd:element name="anything" type="xsd:anyType"/>

Содержание элемента, объявленного этим способом никак не ограничивается.

Фактически, anyType - это тип, задаваемый по умолчанию.

Так что вышеуказанное объявление может выглядеть следующим образом:

<xsd:element name="anything"/>

Аннотации

Язык XML-схемы обеспечивает несколько элементов предназначенных для аннотации схемы.

Содержимое этих элементов предназначено как для чтения человеком, так и для чтения приложением.

Элемент **documentation** предназначен для размещения информации для чтения документа человеком.

Для указания языка комментариев следует использовать атрибут **xml:lang** со всеми элементами documentation.

Элемент **appInfo**, может использоваться, чтобы предоставить информацию для инструментальных средств, таблиц стилей и других приложений.

Элемент **annotation** обычно размещают в начале большинства схем.

Создание моделей содержимого

Все определения комплексных типов представляют собой последовательность объявлений элементов, которые должны появиться в документе-образце.

XML-схема также может обеспечить ограничения вхождения группы элементов в данную модель содержимого.

XML-схема позволяет определить поименованную группу элементов, которые могут использоваться в моделях содержимого комплексного типа.

Также может быть определена непоименованная группа элементов, которые вместе с элементами из поименованной группы будут появляться в документе в той же самой последовательности, в которой были объявлены.

Вместе с тем, группы также могут быть спроектированы таким образом, что только один из элементов группы может появиться в документе-образце.

Для иллюстрации вышесказанного в определение **PurchaseOrderType** из схемы заказа на покупку введем две группы, так что заказ сможет содержать либо специальные элементы для указания адреса отправителя и продавца, либо адрес и отправителя и продавца будут задаваться одним и тем же элементом.

Тогда имеем:

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
   <xsd:choice>
    <xsd:group ref="shipAndBill"/>
    <xsd:element name="singleUSAddress"
                                        type="USAddress"/>
   </xsd:choice>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate"  type="xsd:date"/>
</xsd:complexType>
```

```
<xsd:group id="shipAndBill">
  <xsd:sequence>
    <xsd:element name="shipTo"  type="USAddress"/>
    <xsd:element name="billTo"    type="USAddress"/>
  </xsd:sequence>
</xsd:group>
```

Элемент выбора в группе **choice** обеспечивает правило, по которому в документе-образце может появиться только один из его дочерних элементов.

Элемент **choice** имеет двух потомков.

Один из его потомков - элемент group, который ссылается на поименованную группу shipAndBill, и состоит из последовательности элементов shipTo, billTo.

Второй потомок - singleUSAddress.

Для ограничения появления элементов в группе существует еще одна возможность.

Предположим, что все элементы группы должны появиться один раз или не должны появиться ни разу, причем появляться они могут в произвольном порядке.

Групповой элемент **all** ограничивает модель содержимого сверху.

Кроме того, все дочерние элементы группы должны быть индивидуальными элементами (не группами), и все элементы должны появиться не более одного раза.

То есть это соответствует значениям **minOccurs = 0** и **maxOccurs = 1**.

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
   <xsd:element name="shipTo"  type="USAddress"/>
   <xsd:element name="billTo"   type="USAddress"/>
   <xsd:element ref="comment" minOccurs="0"/>
   <xsd:element name="items" type="Items"/>
  </xsd:all>
 <xsd:attribute name="orderDate"  type="xsd:date"/>
</xsd:complexType>
```

В соответствии с этим определением элемент **comment** может появиться в любом месте **purchaseOrder**, причем как до, так и после элементов **shipTo**, **billTo** или **Items**.

Но при этом он может появиться только однажды.

Кроме того соглашения группы all не позволяют объявлять элементы вроде **comment** вне группы, что ограничивает возможность его использования для многократного появления.

Группы атрибутов

Предположим, что необходимо обеспечить подробную информацию о каждом продукте в заказе на закупку.

Например, вес каждого продукта и предпочтительный вариант отгрузки.

Этого можно достигнуть, добавив к определению типа item (анонимному) объявления атрибутов weightKg и shipBy.

Рассмотрим пример:

```
.....
<xsd:attribute name="weightKg"  type="xsd:decimal"/>
<xsd:attribute name="shipBy">
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
     <xsd:enumeration value="air"/>
     <xsd:enumeration value="land"/>
     <xsd:enumeration value="any"/>
   </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
```

Вместо этого можно создать поименованную группу атрибутов, содержащую все желательные атрибуты элемента **item**, и в объявлении **item** сделать ссылку на эту группу.

```
……
<xsd:attributeGroup ref="ItemDelivery"/>
……
<xsd:attributeGroup id="ItemDelivery">
  <xsd:attribute name="partNum" type="SKU"  use="required"/>
  <xsd:attribute name="weightKg" type="xsd:decimal"/>
  <xsd:attribute name="shipBy">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="air"/>
        <xsd:enumeration value="land"/>
        <xsd:enumeration value="any"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```

Значения Nil

Один из объектов в заказе на закупку, перечисленных в po.xml, **Lawnmower**, не имеет элемента **shipDate**.

Но вообще, отсутствие элемента не дает какой-либо определенной информации.

Это может указывать на то, что информация отсутствует, или не соответствует действительности, или элемент может отсутствовать по другой причине.

Иногда желательно представить не отгруженное изделие, неизвестную или неподходящую информацию явно с помощью элемента, а не отсутствующим элементом.

Для индикации возможности пустого значения элемента Nil-механизм XML-схемы использует специальный признак.

Другими словами, возможное пустое значение элемента обозначается не с помощью какого-либо специального Nil-значения содержимого, а с помощью специального атрибута, показывающего возможность пустого значение элемента.

Рассмотрим пример:

```
<xsd:element name="shipDate" type="xsd:date"
                                    nillable="true"/>
```

Для того чтобы явно указать в документе, что shipDate имеет пустое значение, устанавливаем атрибут nil равным true.

<shipDate xsi:nil="true"></shipDate>

Атрибут nil определен в именном пространстве языка XML-схемы, http://www.w3.org/2001/XMLSchema-instance, и поэтому в документе-образце используется с префиксом (таким как xsi:), связанным с этим именным пространством.

Как и xsd:, префикс xsi: используется в соответствии со стандартным соглашением.