МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

С.В. Одиночкина

Основы технологий XML

Учебное пособие

Санкт-Петербург

2013

УДК 004.655, 004.657, 004.62

С.В. Одиночкина

Основы технологий XML - СПб: НИУ ИТМО, 2013. – 56 с.

В пособии представлено руководство по основным принципам использования языка разметки XML и связанным технологиям по дисциплине “Создание программного обеспечения инфокоммуникационных систем”.

Предназначено для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи.

Рекомендовано к печати Ученым советом факультета инфокоммуникационных технологий, протокол № 3 от 19 марта 2013 г.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Оглавление

Введение

Курс предназначен для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи. Основной целью данной дисциплины является изучение принципов реализации работы с данными с помощью технологии XML в программных приложениях, а также рассмотрение технологий обработки, форматирования и преобразования данных в формате XML.

В результате курса, проводимого под руководством преподавателя, студенты смогут:

- Разрабатывать XML-документы;
- Реализовывать обработку данных в формате XML, в том числе при разработке программных приложений в среде Microsoft .NET Framework;
- Создавать DTD и XML-схемы для реализации возможности валидации XML-документов;
- Использовать возможности XPath при работе со структурой XML-документов;
- Осуществлять форматирование и преобразование документов в формате XML с помощью языка XSLT;

Для прохождения данной дисциплины студенты должны иметь базовые навыки разработки программных продуктов на языке программирования C#.

1. XML и платформа Microsoft .NET Framework

1.1. Расширяемый язык разметки XML

XML (англ. eXtensible Markup Language, расширяемый язык разметки) представляет собой язык разметки, который был создан для описания данных. Язык разметки представляет собой набор символов или последовательностей, вставляемых в текст для передачи информации о его выводе или строении. Таким образом, текстовый документ, размеченный с помощью такого языка, содержит не только сам текст, но и дополнительную информацию о его структуре. Кроме того, язык разметки позволяет вставлять в документ интерактивные элементы и содержание других документов. Разметка разделяется на стилистическую разметку, структурную и семантическую: стилистическая разметка отвечает за внешний вид документа, структурная разметка задает структуру документа, семантическая позволяет описать логику представления данных. XML представляет собой подмножество метаязыка SGML, разработанное для упрощения процесса машинного разбора документа. Можно сказать, что он сам является метаязыком, т.к. не ограничивается набором определенных тегов и используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. XML-документ обычно состоит из процессинговых инструкций, элементов, атрибутов, сущностей и комментариев. Пример синтаксиса XML:

```
<?xml version = “1.0” encoding = “utf-8”?>

<book>
        <title>XSLT</title>
        <author>Тидуэлл Дуг</author>
        <year>2010</year>
        <pages>959</pages>
</book>
```

Обработка XML-документа обычно происходит с использованием таких технологий как DTD/XDR/XSD схем, XPath, XSLT и других.

1.2. Обзор платформы .NET Framework

Платформа .NET Framework — это один из компонентов системы Microsoft Windows. Он позволяет создавать и использовать приложения нового поколения. Назначение платформы .NET Framework:

- создание целостной объектно-ориентированной среды программирования допускающей различные варианты реализации: код может храниться и выполняться локально; выполняться локально, а распространяться через Интернет; или выполняться удаленно;
- предоставление среды выполнения кода, в которой число конфликтов при развертывании программного обеспечения и управлении версиями будет сведено к минимуму;
- обеспечение безопасности выполнения кода в среде — в том числе кода, созданного неизвестным разработчиком или разработчиком с частичным доверием;
- предоставление среды выполнения кода, позволяющей устранить проблемы, связанные с производительностью сред на основе сценариев или интерпретации;
- унификация работы разработчиков в совершенно разных приложениях: как в приложениях Windows, так и в веб-приложениях;
- использование промышленных стандартов во всех областях обмена данными и, как следствие, обеспечения совместимости кода, созданного в .NET Framework, с другими программами.

Платформа .NET Framework состоит из двух основных компонентов: среды CLR и библиотеки классов .NET Framework. Среда CLR — это фундамент платформы .NET Framework. Это своеобразный агент, управляющий кодом во время его выполнения, предоставляющий ключевые службы, связанные с такими процессами, как управление памятью, потоками и удаленными операциями, а также обеспечивающий безопасность типов и другими способами контролирующий правильность кода, гарантируя безопасность и стабильность приложений. Понятие управления кодом является для среды основополагающим. Код, созданный для среды, называется управляемым. Любой другой код называется неуправляемым кодом. Библиотека классов, второй основной компонент платформы .NET Framework, является обширным объектно-ориентированным набором типов, которые можно использовать для разработки самых различных приложений — от классических приложений с интерфейсом командной строки или графическим интерфейсом пользователя до новейших приложений на базе технологий ASP.NET, например веб-форм и веб-служб XML.

Платформа .NET Framework может располагаться на неуправляемом компоненте, который загружает среду CLR в собственные процессы и инициирует выполнение управляемого кода — тем самым создавая среду приложений, в которой может выполняться как управляемый, так и неуправляемый код. Платформа .NET Framework сама предоставляет несколько хост-приложений и поддерживает хост-приложения сторонних разработчиков.

1.3. Язык XML и платформа .NET

Компания Microsoft была одной из первых организаций, которая стала активно использовать XML в качестве одной из ведущих технологий для обмена данными. В рамках ее деятельности была создана группа технологий, поддерживающих различные API для работы с форматом XML в зависимости от языка разработки:

- в случае разработки на C #, Visual Basic, J #, Managed C + + и т.д. используется System.Xml и / или LINQ для XML;
- при написании кода на Visual Basic 6, C, C++ или скриптовых языках – необходимо использовать MSXML библиотеки.
- В случае разработки машинного кода необходимо обращаться к XmlLite API .

Для работы с XML в инструментальной среде Microsoft Visual Studio представлены такие инструменты как XML Editor, XML Schema Designer, XSLT Profiler и другие. В частности, XML Editor обладает встроенной проверкой синтаксиса и автоматической проверкой на соответствие схеме во время ввода, а также возможностями IntelliSense. Кроме того, реализована возможность автоматической генерации схем документа, выполнение XSLT трансформаций, работа с обозревателем XML-схем XML Schema Explorer и другие возможности.

Контрольные вопросы

1. В связи с чем язык XML получил широкое распространение?
2. Что представляет собой язык разметки?
3. Какими особенностями обладает язык разметки XML?
4. Что престав лет собой платформа Microsoft .NET Framework?
5. Каково назначение платформы Microsoft .NET Framework?

6. Из каких компонентов состоит платформа Microsoft .NET Framework?
7. Каковы функции среды CLR?
8. Сто представляет собой библиотека классов Microsoft .NET Framework?
9. На основе каких стандартов строится web-служба XML в рамках платформы Microsoft .NET Framework?
10. Какие API для реализации работы с XML разработаны в рамках платформы Microsoft .NET Framework?

2. Основы XML

2.1. Структура XML-документа

Язык XML был создан для хранения, транспортировки и обмена данными, с его помощью можно реализовать обмен данными между различными системами. XML документ состоит из частей, называемых элементами. Элементы составляют основу XML-документов. Они образуют структуры, которые можно обрабатывать программно или с помощью таблиц стилей. Элементы размечают именованные разделы информации. Элементы строятся с помощью тегов разметки, обозначающих имя, начало и конец элемента. Элементы могут быть вложены друг в друга, на верхнем уровне находится элемент, называемый элементом документа или корневым элементом в котором содержатся остальные элементы. Например:

```
<?xml version="1.0"?>
<planets>
        <planet ID="1">
                <name>Mercury</name>
        </planet>
        <planet ID="2">
                <name>Venus</name>
        </planet>
<!-- There are more planets. -->
</planets>
```

Документ XML может располагаться в одном или нескольких файлах, причем некоторые из них могут находиться на разных машинах. В XML используется специальная разметка для интеграции содержимого разных файлов в один объект, который можно охарактеризовать как

логическую структуру. Благодаря тому, что документ не ограничен одним файлом, XML позволяет создавать документ из частей, которые могут располагаться где угодно.

Документ XML обычно содержит следующие разделы:

- XML-декларация;
- Пролог;
- Элементы;
- Атрибуты;
- Комментарии.

2.2. Декларация XML-документа

XML-декларация обычно находится в первой строке XML-документа. XML-декларация не является обязательной. Однако, если она существует, она должна располагаться в первой строке документа, и до нее не должно быть больше ничего, в том числе пробелов. XML-декларация в схеме документа состоит из следующих элементов:

- Номер версии:
  <?xml version="1.0"?>.
  Это обязательный аргумент. Текущая версия — 1.0.

- Декларация кодировки:
  <?xml version="1.0" encoding="UTF-8"?>
  Это необязательный параметр. Если он используется, то декларация кодировки должна располагаться сразу после информации о версии в XML-декларации. Декларация кодировки должна содержать значение, представляющее собой существующую кодировку символов.

- Декларация автономности, например:
  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>.
  Декларация автономности, как и декларация кодировки, необязательны. Если декларация автономности используется, то она должна стоять на последнем месте в XML-декларации.

2.3. Пролог XML-документа

Прологом называются данные, расположенные после открывающего тега документа или после корневого элемента. Он включает сведения, относящиеся к документу в целом — кодировка символов, структура документа, таблицы стилей.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="book.xsl"?>
<!DOCTYPE book SYSTEM "schema.dtd">
<!--Some comments-->
```

В языке XML есть возможность включения в документ инструкций, которые несут определенную информацию для приложений, которые будут обрабатывать тот или иной документ. Инструкции по обработке в XML создаются следующим образом.

```
<? Приложение Содержимое ?>
```

В XML инструкции по обработке заключаются в угловые кавычки со знаком вопроса. В первой части процессинговой инструкции определяется приложение или система, которой предназначена вторая часть этой инструкции или ее содержимое. При этом инструкции по обработке действительны только для тех приложений, которым они адресованы. Примером процессинговой инструкции может быть следующая инструкция:

```
<?serv cache-document?>
```

2.4.Элементы XML-документа

Элементы в XML документе отвечают за организацию информации и являются основными структурными единицами языка XML. Элементы оформляются следующим образом:

```
<ElementName> Содержимое элемента </ElementName>
```

Теги устанавливают границы вокруг содержимого элемента, если таковое имеется.

У каждого элемента должно быть имя. Имена XML-элементов должны подчиняться следующим правилам:

- Названия могут содержать буквы, цифры и другие символы;
- Названия не могут начинаться с цифры или знака препинания;
- Названия не могут начинаться с букв xml;
- В названии не должно быть пробелов.
- Нельзя допускать пробелов у кавычек (<);
- Имена элементов являются регистрозависимыми;
- Все элементы должны иметь закрывающий тэг.

В XML элементы могут быть двух типов – пустые и непустые. Пустые элементы не содержат в себе никаких данных, таких как текст или другие конструкции, и могут сокращенно записываться следующим образом:

```
<ElementName  />
```

В XML документе обязательно должен присутствовать единственный корневой элемент, все остальные элементы являются дочерними по отношению к единственному корневому элементу. При этом должен строго соблюдаться порядок вложенности элементов:

```
<person>
    <givenName>Peter</givenName>
    <familyName>Kress</familyName>
</person>
```

В данном случае элемент <person> содержит два других элемента, <givenName> и <familyName>. Элемент <givenName> содержит текст Peter, а элемент <familyName> — текст Kress.

2.5.Атрибуты XML-документа

В XML элементы могут содержать атрибуты с присвоенными им значениями, которые помещаются в одинарные или двойные кавычки. Атрибуты позволяют добавлять сведения об элементе с помощью пар «имя-значение». Атрибуты часто используются для определения тех свойств элементов, которые не считаются содержимым элемента, хотя в некоторых случаях (например, HTML-элемент img) содержимое элемента определяется значениями атрибута. Атрибуты могут отображаться в открывающих или пустых тегах, но не в закрывающих тегах. Атрибут для элемента задается следующим образом:

<myElement attribute="value" ></myElement>

Синтаксические правила создания атрибута:

- Декларируются в открывающем тэге;
- Количество атрибутов не ограничено;
- Несколько атрибутов разделяются пробелами;
- Атрибут состоит из имени и значения
    - Каждое имя должно быть уникально в рамках одного элемента;
    - Нельзя использовать пробелы в именах атрибутов;
    - Значение атрибута должно быть в кавычках.

Значение атрибутов может заключаться как в одинарные, так и в двойные кавычки. Возможно также использование одних кавычек внутри других, например:

<myElement attribute="value" > </myElement>
<myElement attribute='value' ></myElement>
<myElement attribute='"value"' ></myElement>

2.6. Комментарии XML-документа

Содержимое, не предназначенное для синтаксического анализа (например, замечания о структуре документа или редактировании), можно заключить в комментарии. Комментарии начинаются с группы символов <!-- и заканчиваются группой символов -->:

<!—Текст комментария -->

Комментарии могут находиться в прологе документа, в том числе в определении типа документа (DTD), после документа и в текстовом содержимом. Комментарии не могут находиться внутри значений атрибутов. Они также не могут находиться внутри тегов.

Синтаксический анализатор считает концом комментария символ >; после этого символа он возобновляет обработку XML-документа в обычном режиме. Поэтому строка > не может находиться внутри комментария. За исключением этого, в комментариях могут использоваться любые допустимые символы XML. Поэтому комментарии очень полезны, если нужно убрать комментарий из области обработки

синтаксического анализатора, не удаляя само содержимое из документа. Для временного удаления разметки можно использовать следующие комментарии:

<!--- <test pattern="SECAM" /><test pattern="NTSC" /> --><!--Текст комментария -->

При создании комментария необходимо учитывать следующее:

1.В тексте комментария не может быть двух символов «-» подряд.
2.Комментарий не может заканчиваться символом «-».

2.7. Текстовые данные XML-документа

Благодаря поддержке набора символов Юникода XML поддерживает целый диапазон символов, в том числе буквы, цифры, знаки препинания и другие символы. Большинство управляющих символов и символы совместимости Юникода не допускаются.

Весь текст XML-документа анализируется парсером, т.е. является парсируемым (PCDATA), но в случае необходимости определенные разделы можно «скрыть» для проверки и они будут игнорироваться парсером (т.н. непарсируемые данные, CDATA). Разделы CDATA дают возможность сообщить средству синтаксического анализа, что среди символов, содержащихся в разделе CDATA, отсутствует разметка. Это упрощает создание документов с разделами, в которых могут появиться отдельные символы разметки, но на самом деле разметки нет. В разделы CDATA часто помещают содержимое на языке сценариев, а также образцы содержимого XML и HTML.

Раздел CDATA в схеме документа использует следующую конструкцию:

<![CDATA[Some information using <, >,]]>

При обнаружении начального тега <![CDATA[, средство синтаксического анализа XML рассматривает все последующее содержимое как символы, не пытаясь интерпретировать их как разметку элемента или сущности. Ссылки на символы в разделах CDATA не работают. Обнаружив завершающий тег]]>, средство синтаксического анализа возобновляет нормальный синтаксический анализ. Например, в XML-документ можно включить любой из приведенных далее разделов

CDATA, и средство синтаксического анализа не воспримет их как ошибочные.

```
<![CDATA[</this is malformed!</malformed</malformed & worse>]]>
```

или

```
<script>
        <![CDATA[
                function matchwo(a,b)
                        {
                                if (a < b && a < 0) then
                                  {
                                        return 1;
                                  }
                                      else
                                        {
                                                return 0
                                        }
                                }
        ]]>
</script>
```

Содержимое разделов CDATA может содержать только символы, разрешенные для содержимого XML; нельзя экранировать таким образом управляющие символы и символы совместимости. Кроме того, в раздел CDATA не может входить последовательность]]>, поскольку эти символы означают завершение раздела. Это значит, что разделы CDATA не могут быть вложенными друг в друга. Эта последовательность также используется в некоторых сценариях. В сценариях обычно можно вместо последовательности]]> использовать]]>.

Помимо непарсируемых данных в XML-тексте можно использовать ссылки на символы и сущности. Ссылки на символы и сущности позволяют включать данные в XML-документ, ссылаясь на них, вместо того чтобы вводить символы в документ напрямую. Такой способ удобно применять в следующих ситуациях:

- Символы нельзя ввести в документ напрямую, так как они будут интерпретироваться как разметка.
- Символы нельзя ввести в документ напрямую из-за ограничений устройства ввода.
- Невозможна надежная передача символов через процессор, ограниченный однобайтными символами.

- Символьная строка или фрагмент документа часто повторяются и могут быть сокращены.

Для представления содержимого в XML используются числовые или синтаксические конструкции, начинающиеся с символа амперсанда (&) и заканчивающиеся точкой с запятой (;). Ссылки на символы позволяют вставлять символы Юникода, для которых в качестве кодовой точки Юникода задан числовой код. Кодовые точки можно задавать либо в десятичном, либо в шестнадцатеричном представлении:

& #value; - синтаксис, используемый для десятичных ссылок.
&# xvalue; - синтаксис, используемый для шестнадцатеричных ссылок.

Например, чтобы вставить символ евро, который до сих пор отсутствует на многих клавиатурах, можно вставить в документ ссылку € или €.

В приведенной таблице 1 перечислены пять встроенных сущностей для символов, используемых в XML-разметке:

Таблица 1. Встроенные сущности XML

Сущность	**Ссылка на сущность**	**Значение**
lt	<	< (меньше чем)
gt	>	> (больше чем)
amp	&	& (амперсанд)
apos	'	' (апостроф или одиночная кавычка)
quot	"	" (двойная кавычка)

В ситуациях, когда символ может привести к ошибочной интерпретации структуры документа средством синтаксического анализа XML, используйте сущность вместо ввода символа. Ссылки на сущности ' и " чаще всего используются в значениях атрибутов. Например, чтобы написать «Me&You», используйте Me&You. Чтобы написать «a<b», используйте a<b. Чтобы написать «b>c», используйте b>c.

Можно определить собственные сущности, как HTML определяет набор сущностей для использования в HTML. Значение &ap не распознается как HTML-файл. Для преобразования в HTML нужно использовать $#.....

При работе с определением типа документа (DTD), определяющим сущности, можно ссылаться на эти сущности в содержимом документа, используя следующий синтаксис:

&entityName;

При обработке пробелов в документе XML необходимо учитывать, что в спецификации XML, опубликованной консорциумом World Wide Web (W3C), различные способы обозначения конца строки приведены к единому способу обозначения, однако при этом сохраняются все остальные пробелы, за исключением пробелов в значениях атрибутов. Кроме того, в XML предусмотрен набор средств, с помощью которых документы могут сообщать приложениям, следует ли сохранять пробелы. Согласно существующему стандарту XML 1.0 пробелы перед XML-декларацией не допускаются.

```
<?xml version="1.0"?>
 <root>
        <element>something</element>
 </root>
```

Если перед XML-декларацией есть пробелы, они будут рассматриваться как инструкция по обработке. Информация (в частности, о кодировке) не обязательно используется средством синтаксического анализа.

Средства синтаксического анализа XML обязательно сообщают обо всех пробелах, найденных в содержимом элементов внутри документа. Поэтому следующие три документа с точки зрения средства синтаксического анализа XML будут различными:

Документ 1:

```
<root>
        <data>1</data>
        <data>2</data>
        <data>3</data>
</root>
```

Документ 2:

```
<root><data>1</data><data>2</data><data>3</data></root>
```

Документ 3:

```
<root>  <data>1</data>  <data>2</data>  <data>3</data>  </root>
```

Для XML-приложений, ориентированных на документы, пробелы могут быть чрезвычайно важны.

Авторы документов могут использовать атрибут xml:space для обозначения частей документов, где пробелы представляют определенную важность. Атрибут xml:space также может использоваться в таблицах стилей, обеспечивая сохранение пробелов в представлении документа. Однако поскольку многие XML-приложения не распознают атрибут xml:space, его использование может быть лишь рекомендацией.

Атрибут xml:space может принимать два значения:

- default - это значение позволяет приложению обрабатывать пробелы при необходимости. Если не включить атрибут xml:space, то результат будет такой же, как при использовании значения default.

- preserve - это значение показывает приложению, что пробелы следует сохранить в неизменном виде, так как они могут представлять важность.

Значения атрибутов xml:space применяются ко всем потомкам содержащего атрибут элемента, кроме случаев, когда значение переопределяется в одном из дочерних элементов.

Например, в следующих документах относительно пробелов задано одинаковое поведение:

Документ 1:

```
<poem xml:space="default">
	<author>
		<givenName>Alix</givenName>
		<familyName>Krakowski</familyName>
	</author>
	<verse xml:space="preserve">
		<line>Roses  are  red,</line>
		<line>Violets  are  blue.</line>
		<signature xml:space="default">-Alix</signature>
	</verse>
</poem>
```

Документ 2:

```
<poem xml:space="default">
	<author xml:space="default">
```

```
        <givenName xml:space="default">Alix</givenName>
        <familyName xml:space="default">Krakowski</familyName>
    </author>
    <verse xml:space="preserve">
        <line xml:space="preserve">Roses  are  red,</line>
        <line xml:space="preserve">Violets  are  blue.</line>
        <signature xml:space="default">-Alix</signature>
    </verse>
</poem>
```

В обоих примерах приложение получает сообщение, что все пробелы в строках стихотворения следует сохранить, но пробелы в других частях документа обрабатываются по необходимости.

По умолчанию службы Microsoft XML Core Services (MSXML) не обрабатывают атрибут xml:space. Если приложение должно учитывать атрибут xml:space, то перед синтаксическим анализом для свойства preserveWhiteSpace объекта DOMDocument следует задать значение True.

```
xmldoc= new ActiveXObject("Msxml2.DOMDocument.5.0");

xmldoc.preserveWhiteSpace = true;

xmldoc.load(url);
```

В службах MSXML также предусмотрены параметры, которые позволяют перепоручить обработку пробелов в приложении средству синтаксического анализа. Дополнительные сведения см. в разделе «White Space and the DOM» (пробелы и модель DOM) документации по пакету MSXML SDK.

Приложения по обработке XML сохраняют все пробелы в содержимом элементов, но часто нормализуют пробелы в значениях атрибутов. Символы табуляции, символы возврата каретки и пробелы обрабатываются как единичные пробелы. В определенных типах атрибутов удаляются пробелы, находящиеся до или после тела значения. Повторяющиеся пробелы внутри значения заменяются на один пробел. (При наличии определения DTD эта обработка проводится для всех атрибутов, не принадлежащих к типу CDATA). Например, XML-документ может содержать следующие данные:

```
<whiteSpaceLoss note1="this is a note." note2="this
is
a
note.">
```

Средство синтаксического анализа XML передаст оба значения атрибута в виде "this is a note." с преобразованием разрывов строк в одиночные пробелы.

Средства синтаксического анализа XML обрабатывают сочетание символов возврата каретки и перевода строки (CRLF) как одиночные символы возврата каретки или перевода строки. Все они передаются как единичные символы перевода строки (LF). При сохранении документов приложения могут использовать соответствующие соглашения об обработке конца строки.

2.8.Пространства имен XML-документа

Поскольку имена элементов в XML не фиксированы, часто случаются конфликты, когда два различных документа используют одно и то же имя для описания двух различных типов элементов, например:

```
<Order>
    <Employee>
        <Name>Jane Doe</Name>
        <Title>Developer</Title>
    </Employee>
    <Product>
        <Title>The Joshua Tree</Title>
        <Artist>U2</Artist>
    </Product>
</Order>
```

Пространства имен XML дают возможность избежать конфликтов имен элементов. Они задаются при помощи уникальных идентификаторов URI. Чтобы упростить работу, были разработаны специальные префиксы пространств имен, которые позволяют с легкостью определить, какой схеме принадлежит тот или иной элемент документа. Общий синтаксис:

```
<ElementName xmlns:Prefix="http://contoso.msft/namespace_for_examples">
    <Prefix:AnyElement>Some Data</Prefix:AnyElement>
    <AnotherElement>More Data</AnotherElement>
```

Например:

```
<Order xmlns:hr="http://hrweb" xmlns:mkt="http://market">
```

```
        <hr:Name>Jane Doe</hr:Name>
        <hr:Title>Developer</hr:Title>
        <mkt:Title>The Joshua Tree</mkt:Title>
        <mkt:Artist>U2</mkt:Artist>
</Order>
```

Префиксы пространств имен задаются как атрибуты с именами, которые начинаются последовательностью xmlns. Созданный префикс пространств имен может использоваться только в собственном имени и во вложенных элементах, но не за пределами элемента в котором он был создан. Сами префиксы не относят элемент к той или иной схеме. Эту функцию выполняют уникальные идентификаторы, которые поставлены в соответствие этим префиксам. Таким образом, два элемента с разными префиксами которым заданы одинаковые идентификаторы будут считаться принадлежащими к одной схеме.

Существует еще один способ, который позволяет не проставлять префиксы в элементах. Для этого достаточно задать пространство имен по умолчанию. В этом случае все вложенные элементы будут принадлежать пространству имен родительского элемента. При этом не теряется возможность использовать другие пространства имен для дочерних элементов. Для этого достаточно вручную прописать нужное пространство имен, используя атрибут xmlns.

```
<Order>
<Employee xmlns="http://hrweb">
            <Name>Jane Doe</Name>
            <Title>Developer</Title>
    </Employee>
    <Product xmlns="http://market">
            <Title>The Joshua Tree</Title>
            <Artist>U2</Artist>
    </Product>
</Order>
```

В представленном примере прослеживается так называемое наследование. Если для элемента не указано пространство имен, то ему автоматически присваивается пространство имен ближайшего родительского элемента.

Обычно вышеприведенный способ не очень популярный и чаще всего используются префиксы пространств имен. Но в некоторых случаях их можно опустить и использовать способ с заданием пространства имен по умолчанию. Чтобы не оставалось никаких вопросов рассмотрим аналогичный способ представления последнего примера.

Контрольные вопросы

1.
2. Что представляет собой XML-документ?
3. Из каких разделов состоит XML-документ?
4. Какую информацию содержит декларация XML-документа?
5. Какая информация размещается в прологе XML-документа?
6. Как строятся элементы XML-документа?
7. Каковы синтаксические правила создания атрибутов XML-документа?
8. Для чего в XML-документах используются комментарии?
9. Чем парсируемые данные в XML-документе отличаются от непарсируемых данных?
10. Что представляют собой сущности XML-документа?
11. Для чего в XML-документе необходимо пространство имен?

3. Технологии обработки данных в формате XML

Данные, представленные в формате XML, обычно проходят процедуру валидации, т.е. проверки грамматики документа на соответствие определенным схемам. В таких схемах находится описание структур данных XML-документа. Необходимость проверки грамматики XML-документов заключается в следующем:

- XML-документ может быть предназначен для другой системы;
- XML-документ может содержать некорректные данные;
- XML-документ может содержать ошибки в структуре.

При обработке XML данных валидация – это фундамент для дальнейших действий с XML-документом, информация в валидном XML-документе может быть отправлена да дальнейшую обработку в целевой модуль. Существуют три основные разновидности схем: DTD, XDR и XML-схемы (XSD). На сегодняшний день актуальными являются DTD схемы и более современный подход – XML-схемы (XSD).

3.1. Схема DTD

DTD (Document Type Definition, определение типа документа) - это язык описания структуры XML-документа, который используется для проверки грамматики XML-документа и его соответствия определенному

стандарту. Это позволяет парсеру на этапе обработки определить, соответствует ли документ необходимым требованиям, т.е. является ли документ валидным. DTD описывает:

- Какие элементы могут присутствовать в документе;
- Вхождение элементов (повторения и т.п.);
- Возможные атрибуты элементов;
- Обязательные / необязательные атрибуты;
- PCDATA и CDATA;
- Применяемые в документе сущности.

Например:

```
<?xml version="1.0"?>
	<!DOCTYPE note [
		<!ELEMENT note (to, from, Sbj, msg)>
		<!ELEMENT to     (#PCDATA)>
		<!ELEMENT from   (#PCDATA)>
		<!ELEMENT Sbj (#PCDATA)>
		<!ELEMENT msg  (#PCDATA)>
]>
<note>
	<to>Sunny</to>
	<from>Oliver</from>
	<Sbj>Hello</Sbj>
	<msg>This is a good day!</msg>
</note>
```

В рамках DTD доступны четыре определяющих инструкций для разработки определения типа документа:

- ATTLIST (список атрибутов) - объявляет список XML-атрибутов. Эти атрибуты определяются именем, типом данных, неявными значениями по умолчанию и именами любых элементов, позволяющих их использование.
- ELEMENT - объявляет имя типа XML-элемента и его допустимые вложенные (дочерние) элементы.
- ENTITY - объявляет специальные символьные ссылки, текстовые макросы (наподобие инструкции #define языка C/C++) и другое повторяющееся содержимое (наподобие инструкции #include языка C/C++).

- NOTATION - объявляет внешнее содержимое, не относящееся к XML (например, двоичные графические данные), а также внешнее приложение, которое обрабатывает это содержимое.

3.1.1. Инструкция ATTLIST

Инструкция ATTLIST используется для перечисления и объявления всех атрибутов, которые могут принадлежать элементу. Сначала указывается имя элемента (или элементов), к которому относится список атрибутов. Затем для всех атрибутов по очереди указывается имя, обязательность и символьные данные, допустимые в качестве значения.

Синтаксис инструкции:

<!ATTLIST elementName attributeName dataType default >

Возможные параметры:

- elementName - имя элемента, к которому относится список атрибутов.
- attributeName - имя атрибута. Этот параметр повторяется столько раз, сколько нужно для перечисления всех атрибутов, используемых с elementName.
- dataType - тип данных для атрибута, названного в параметре attributeName; должен иметь одно из следующих значений:
    - CDATA – атрибут содержит только символьные данные.
    - ID - значение атрибута должно быть уникальным. Оно не может повторяться в других элементах или атрибутах данного документа.
    - IDREF – атрибут ссылается на значение другого атрибута типа ID из данного документа.
    - ENTITY – значение атрибута должно соответствовать имени внешней сущности ENTITY, не подвергавшейся синтаксическому разбору и объявленной в том же определении DTD.
    - ENTITIES - значение атрибута содержит несколько имен внешних сущностей, не подвергавшихся синтаксическому разбору и объявленных в том же определении DTD.
    - NMTOKEN – значение атрибута должно быть лексемой имени. Токены имени допускают символьные значения данных, но накладывают больше ограничений, чем тип CDATA. Лексема имени может содержать буквы, цифры и некоторые знаки препинания — точки, тире, символы

подчеркивания и двоеточия. Однако значения лексем имени не могут содержать никаких пробелов и приравненных к ним символов.

- Enumerated – Значения атрибута ограничены перечисленными в списке.

- default - значение по умолчанию для атрибута, заданного параметром attributeName. Возможные значения по умолчанию приведены в таблице 2:

Таблица 2. Значения по умолчанию атрибута default

Значения по умолчанию	**Описание**
#REQUIRED	Атрибут должен присутствовать в XML-документе, иначе при синтаксическом разборе будет сформирована ошибка. В некоторых случаях, чтобы избежать возникновения ошибки, можно по желанию использовать поле defaultValue, поместив его непосредственно за этим ключевым словом.
#IMPLIED	Атрибут может присутствовать в XML-документе, но его отсутствие не вызывает ошибки при синтаксическом разборе. В некоторых случаях можно по желанию также использовать поле defaultValue, поместив его непосредственно за этим ключевым словом.
#FIXED	Значение атрибута зафиксировано в определении DTD; изменить или переопределить его в XML-документе нельзя. При использовании этого ключевого слова непосредственно за ним нужно обязательно поместить поле defaultValue для объявления постоянного значения атрибута.
defaultValue	Значение по умолчанию, или фиксированное. Синтаксический анализатор вставляет это значение в XML-документ, если атрибут отсутствует или не используется в данном XML-документе. Все значения должны быть заключены в одинарные или двойные кавычки.

Например:

XML-документ:
<price currency = “RUR” symbol = “ρ”>
120 </price>

DTD-схема:
<!ATTLIST price
currency CDATA #REQUIRED
symbol CDATA #IMPLIED>

3.1.2. Инструкция ELEMENT

Инструкция ELEMENT применяется для объявления каждого элемента, который используется внутри типа документа, определенного в DTD. Сначала инструкция объявляет имя элемента, а затем определяет, какое содержимое допустимо в элементе. Синтаксис инструкции:

<!ELEMENT name content >

Возможные параметры:

- name - имя элемента. Необходимо точно воспроизвести регистр.
- Content - допустимая модель содержимого для элемента может быть одной из следующих.
  - ANY — внутри элемента допускается любое содержимое. При использовании в объявлении элемента, это ключевое слово разрешает открытие неограниченной модели содержимого для элементов и всех дочерних узлов.
  - EMPTY — в элементе не допускается содержимое, он должен оставаться пустым.
  - Объявленное правило для содержимого — в этом случае требуется написать правило для содержимого, заключенное в круглые скобки.

В таблице 3 показаны зарезервированные ключевые слова и символы пунктуации, которые могут быть использованы вместе с именами других элементов, объявленных в DTD, для конструирования правила для содержимого элемента.

Таблица 3. Зарезервированные слова и символы ELEMENT

Символы	Описание
#PCDATA	Содержимое элемента может быть анализируемыми символьными данными.
name	Имя элемента. Имя определяемого элемента или других

	элементов, определенных в DTD, заданное с помощью дополнительных объявлений ELEMENT. Если в правиле для содержимого нет других символов или знаков пунктуации, то допускается и требуется только одно вхождение именованного элемента.
()	В объявлении модели содержимого для элемента требуются как минимум одни круглые скобки (можно вкладывать дополнительные скобки). Дополнительные скобки могут понадобиться для уточнения более сложной модели содержимого для элемента.
|	Вертикальная черта используется для отделения двух именованных элементов. При использовании она указывает, что любой из элементов (до или после вертикальной черты) может отображаться как дочерний элемент.
,	Запятая используется для отделения двух именованных элементов или вложенных правил. При использовании она указывает на порядок отображения элементов или правил.
?	Вопросительный знак используется в качестве суффикса или операнда. При использовании указывает, что предыдущий элемент или правило является необязательным. Может использоваться только один раз в данном фрагменте структуры XML-документа.
+	Знак плюс используется в качестве суффикса или операнда. При использовании указывает, что предыдущий элемент или правило является обязательным. Может использоваться более одного раза в данном фрагменте структуры XML-документа.
*	Звездочка используется в качестве суффикса или операнда. При использовании указывает, что предыдущий элемент или правило является необязательным. Может использоваться более одного раза в данном фрагменте структуры XML-документа.

XML-документ:

```
<letter>
        <title>Заголовок письма</title>
        <message>Текст письма</message>
</letter>
```

DTD-схема:

<!ELEMENT letter (title, message)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT message (#PCDATA)>

3.1.3. Инструкция ENTITY

Инструкция ENTITY используется для определения сущностей в DTD с целью их использования как в связанном с DTD XML-документе, так и собственно в DTD. ENTITY представляет собой сокращенную запись для размещения в XML-документе. Сокращенное имя указывается для имени параметра. Инструкции ENTITY особенно полезны в ситуациях, когда требуется повторять сведения или использовать объемные текстовые блоки, которые можно хранить в отдельных файлах. В XML-документе за сокращенным именем следует символ «;» (&abbName;)

DTD-схема:

<!ENTITY name “Hello, world!”>

XML-документ:

<element>&name;</element>

3.1.4. Инструкция NOTATION

Инструкция NOTATION используется для определения нотаций. Нотации позволяют XML-документу передавать внешним приложениям уведомляющие сведения. Синтаксис инструкции:

<!NOTATION name [SYSTEM|PUBLIC publicID] resource >

Возможные параметры:

- Name - имя нотации. Обязательно для всех определений нотации.
- publicID - общий идентификатор нотации. Требуется только если в объявлении используется ключевое слово PUBLIC.
- resource - значение для нотации. Обязательно для всех определений нотации. Обычно если нотация является общей, то это идентификатор URI, понятный для человека, но не для компьютеров. Для системных нотаций это значение может указывать имя файла приложения в системе, которое можно использовать для обработки данных, отличных от XML. Например, можно объявить нотацию для помощи в обработке

непроанализированной внешней сущности, например графического файла в формате JPEG или GIF.

```
<!NOTATION MyCatalog SYSTEM "http://example.microsoft.com/catalog">
```

К основным недостаткам DTD схем относят синтаксические отличия от языка XML, а также отсутствие возможностей работы с типами данных.

3.2. XML-схема (XSD)

XML-схема - это основанная на XML современная альтернатива DTD, описывающая структуру XML-документа, в том числе:

- Определяет элементы, которые могут появляться в документе;
- Определяет атрибуты, которые могут появляться в документе;
- Определяет, какие элементы являются дочерними;
- Определяет последовательность, в которой появляются дочерние элементы;
- Определяет число дочерних элементов;
- Определяет пустой ли элемент или он может включать в себя текст;
- Определяет типы данных элементов и атрибутов;
- Определяет значения атрибутов по умолчанию.

Пример XML-схемы:

```
<?xml version = "1.0" encoding = "utf-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
<!-- complexType -->
<xs:element name = "Orders">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name = "Person">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name = "Customer">
                                                        <xs:complexType>
                                                                <xs:sequence>
                                                                        <xs:group ref =
"NewPersonGroup"/>
                                                                        <xs:element name =
"Information" type = "xs:string"/>
                                                                </xs:sequence>
                                                        </xs:complexType>
```

```
                                        </xs:element>

                                        <xs:element name = "ProjectManager">
                                                <xs:complexType>
                                                        <xs:sequence>
                                                                <xs:group ref =
"NewPersonGroup"/>
                                                                <xs:element name =
"Style" type = "xs:string"/>
                                                                <xs:element name =
"Education" type = "xs:string"/>
                                                        </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>

                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <xs:element name = "Project">
                        <xs:complexType>
                                <xs:sequence>
                                        <xs:element name = "Address" type =
"xs:string"/>
                                        <xs:element name = "Area" type =
"xs:decimal"/>
                                        <xs:element name = "Duration" type =
"xs:integer"/>
                                        <xs:element name =
"EmployeesNumber" type = "xs:integer"/>
                                        <xs:element name = "Price" type =
"xs:integer"/>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>

        </xs:sequence>
    </xs:complexType>
</xs:element>

    <xs:group name = "NewPersonGroup">
        <xs:sequence>
            <xs:element name = "Name" type = "xs:string"/>
            <xs:element name = "Phone" type = "xs:string"/>
```

```
                <xs:element name = "City" type = "xs:string"/>
                <xs:element name = "Age" type = "xs:integer"/>
            </xs:sequence>
        </xs:group>
</xs:schema>
```

3.2.1. Элементы XML-схем

Элементы верхнего уровня XML-схем представлены в таблице 4:

Таблица 4. Элементы верхнего уровня XML-схем

Элемент	Описание
<xsd:annotation>	Определяет заметку.
<xsd:attribute>	Объявляет атрибут.
<xsd:attribute>	Группирует набор объявлений атрибутов таким образом, что их можно включить в качестве группы в определения сложных типов.
<xsd:complexType>	Объявляет сложный тип, определяющий набор атрибутов и содержимое элемента.
<xsd:element>	Объявляет элемент.
<xsd:group>	Группирует набор объявлений элементов таким образом, что их можно включить в качестве группы в определения сложных типов
<xsd:import>	Определяет пространство имен, на компоненты схемы которого ссылается содержащая схема.
<xsd:include>	Включает указанный документ схемы в целевое пространство имен содержащей схемы.
<xsd:notation>	Содержит определение нотации, описывающей формат не-XML данных в XML-документе. Определение нотации схемы XML – это видоизменение определений XML 1.0 NOTATION.
<xsd:redefine>	Позволяет переопределить в текущей схеме простые и сложные типы, группы и группы атрибутов, полученные из внешних файлов схем.
<xsd:simpleType>	Объявляет простой тип, который определяет ограничения на значения атрибутов или элементов, включающих только содержимое, а также сведения о них.

3.2.2. Примитивы XML-схем

В таблице 5 приведены элементы, которые могут содержать атрибуты minOccurs и maxOccurs. Такие элементы всегда отображаются как части определения сложного типа, либо как части именованной группы моделей.

Таблица 5. Примитивы XML-схем

Элемент	Описание
<xsd:all>	Позволяет элементам группы появляться (или не появляться) в содержащем элементе в любом порядке.
<xsd:any>	Разрешает любому элементу из указанных пространств имен появляться в содержащем их элементе **sequence** или **choice**.
<xsd:choice>	Позволяет присутствовать в элементе-контейнере единственному элементу выбранной группы.
<xsd:element>	Объявляет элемент.
<xsd:group>	Группирует набор объявлений элементов таким образом, что их можно включить в качестве группы в определения сложных типов
<xsd:sequence>	Требует, чтобы элементы группы появлялись в содержащем их элементе в указанной последовательности.

3.2.3. Атрибуты XML-схем

В таблице 6 приведены элементы, определяющие атрибуты в схемах.

Таблица 6. Элементы, определяющие атрибуты XML-схем

Элемент	Описание
<xsd:anyAttribute>	Разрешает любому элементу из указанных пространств имен появляться в содержащем их элементе **complexType** или **attributeGroup**.
<xsd:attribute>	Объявляет атрибут.
<xsd:attribute>	Группирует набор объявлений атрибутов таким образом, что их можно включить в качестве группы в определения сложных типов.

3.2.4. Простые типы элементов XML-схем

Простыми считаются элементы, которые не содержат других элементов или атрибутов. Синтаксис простых элементов:

```
<xs:element name="name" type="type"/>
```

Например, для XML-документа вида:

```
<note>
    <to>Anny</to>
    <from>Mama</from>
    <title>Hello</title>
    <mess>This is a good day!</mess>
</note>
```

Описание простых элементов в XML-схеме будет следующего вида:

```
<xs:element name="to" type="xs:string"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="mess" type="xs:string"/>
```

В таблице 7 приведены элементы, создающие определения простых типов.

Таблица 7. Определения простых типов XML-схем

Элемент	Описание
<xsd:annotation>	Определяет заметку.
<xsd:appinfo>	Задает сведения, используемые приложениями в элементе **annotation**.
<xsd:documentation>	Задает сведения, которые читают или используют пользователи в элементе **annotation**.
<xsd:element>	Объявляет элемент.
<xsd:list>	Определяет коллекцию из одного определения **simpleType**.
<xsd:restriction> (simpleType)	Задает ограничения на определение **simpleType**.
<xsd:union>	Определяет коллекцию из нескольких определений **simpleType**.

3.2.5. Комплексные типы элементов XML-схем

Комплексные (сложные) типы элементов XML-схем – это элементы, содержащие вложенные элементы или атрибуты. Синтаксис сложных элементов:

```
<xs:element name="name">
  <xs:complexType>
    …
  </xs:complexType>
</xs:element>
```

Например, для XML-документа вида:

```
<note>
        <to>Anny</to>
        <from>Mama</from>
        <title>Hello</title>
        <mess>This is a good day!</mess>
</note>
```

XML-схема будет следующего вида:

```
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="mess" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

В таблице 8 приведены элементы, создающие определения комплексных (сложных) типов.

Таблица 8. Определения сложных типов XML-схем

Элемент	Описание
<xsd:all>	Позволяет элементам группы появляться (или не появляться) в содержащем элементе в любом порядке.
<xsd:annotation>	Определяет заметку.
<xsd:any>	Разрешает любому элементу из указанных пространств имен появляться в

	содержащем их элементе **sequence** или **choice**.
<xsd:anyAttribute>	Разрешает любому атрибуту из указанных пространств имен появляться в содержащем их элементе **complexType** или **attributeGroup**.
<xsd:appinfo>	Задает сведения, используемые приложениями в элементе **annotation**.
<xsd:attribute>	Объявляет атрибут.
<xsd:attribute>	Группирует набор объявлений атрибутов таким образом, что их можно включить в качестве группы в определения сложных типов.
<xsd:choice>	Позволяет присутствовать в элементе-контейнере одномуи только одному элементу выбранной группы.
<xsd:complexContent>	Содержит расширения или ограничения для сложного типа, хранящего смешанное содержимое или только элементы.
<xsd:documentation>	Задает сведения, которые читают или используют пользователи в элементе **annotation**.
<xsd:element>	Объявляет элемент.
<xsd:extension> (simpleContent)	Содержит расширения **simpleContent**. Выполняется расширение простого или сложного типа, содержащего простое содержимое, путем добавления указанных атрибутов, групп атрибутов, либо атрибута **anyAttribute**.
<xsd:extension> (complexContent)	Содержит расширения для **complexContent**.
<xsd:group>	Группирует набор объявлений элементов таким образом, что их можно включить в качестве группы в определения сложных типов
<xsd:restriction> (simpleContent)	Задает ограничения на определение **simpleContent**.
<xsd:restriction> (complexContent)	Задает ограничения на определение **complexContent**.

Контрольные вопросы

1. Для чего необходима валидация XML-документа?
2. Что определяет DTD-схема документа?
3. Какие инструкции используются в DTD-схеме для описания грамматики XML-документа?
4. Каковы достоинства и недостатки DTD-схем?
5. Что представляет собой XML(XDR)-схема?
6. Какие структурные элементы и характеристики XML-документа описывает XML(XDR)-схема?
7. В чем заключается разница между простыми и комплексными типами элементов в XML(XDR)-схемах?
8. Какие типы элементов, описываемых с помощью XML(XDR)-схем, могут содержать атрибуты?
9. Что определяют примитивы XML(XDR)-схем?
10. В чем заключается основная разница между DTD и XML(XDR)-схемами?

4. Форматирование и преобразование документов

Согласно рекомендациям консорциума W3C, с целью форматирования и преобразования XML-документов используется язык преобразований XSLT, который является одним из направлений XSL. XSL (eXtensible Stylesheet Language) представляет собой семейство рекомендаций консорциума W3C, описывающее языки преобразования и визуализации XML-документов. Основные направления XSL:

- XSL Transformations (XSLT) — язык преобразований XML-документов.
- XPath — язык путей и выражений, используемый в XSLT для доступа к отдельным частям XML-документа.

и некоторые другие.

4.1.Язык адресации XPath

XPath представляет собой язык задания путей к элементам XML-документа. Разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. Для адресации частей XML-документа выражение языка XPath использует обозначение пути, похожее на обозначение в URL-адресе. Выполняется оценка выражения для того, чтобы задать объекту один из типов: тип набора узлов, логический, числовой или строковый тип. Например, выражение book/author возвращает набор узлов элементов

<author>, содержащихся в элементах <book>, при условии, что такие элементы объявлены в исходном XML-документе. Кроме того, выражение XPath может содержать предикаты (критерии фильтра) или вызовы функций. Например, выражение book[@type="Fiction"] ссылается на элементы <book>, для которых атрибут type принимает значение "Fiction".

4.1.1.Пути расположения в XPath

Путь расположения представляет собой выражение XPath, которое используется для выбора набора узлов, относящихся к узлу контекста. Результатом оценки выражения пути расположения является набор узлов, содержащий узлы, определенные путем расположения. В путь расположения могут рекурсивно входить выражения, используемые для фильтрации наборов узлов.

Синтаксически путь расположения состоит из одного и более шагов определения расположения, отделенных друг от друга косой чертой (/):

locationstep/locationstep/locationstep

Каждый шаг определения расположения в порядке очереди выбирает набор узлов, относящихся к узлу контекста, т. е. к узлу, выбранному предыдущим шагом определения расположения. Путь расположения, выраженный подобным образом, является относительным путем расположения. Абсолютный путь расположения берет начало от корневого элемента:

/locationstep/locationstep/locationstep

Шаги определения расположения в пути расположения оцениваются слева направо. Крайний левый шаг определения расположения выбирает набор узлов, относящихся к узлу контекста.Эти узлы затем становятся узлами контекста для обработки следующего шага определения расположения. Обработка узлов и смена узла контекста повторяется до тех пор, пока не будут обработаны все шаги определения расположения.

Путь расположения может иметь как полный, так и сокращенный синтаксис. Путь расположения с полным синтаксисом выглядит следующим образом:
ось::узел[предикат]

В этом синтаксисе элемент «ось» определяет, как узлы, выбранные шагом определения расположения, располагаются относительно узла контекста. Элемент «узел» определяет тип узла и развернутое имя узлов, выбранных шагом определения расположения. Элемент «предикат» является критерием фильтра, используемым для дальнейшего уточнения выбора узлов в шаге определения расположения. Предикаты являются необязательными элементами. В сокращенном синтаксисе пути расположения указатель оси (axis::) выражается в шаге определения расположения неявно, вместо этого он описывается набором ярлыков. В таблице 9 приводится список некоторых сокращений

Таблица 9. Список сокращенного синтаксиса путей

Полный синтаксис	Сокращенный синтаксис
child::*	*
attribute::*	@*
/descendant-or-self::node()	//
self::node()	.
parent::node()	..

4.1.2. Оси выборки XPath

Шаг определения расположения задает набор узлов (node-set) по отношению к узлу контекста. Шаг доступа состоит из трех частей: необязательной оси, проверки узла и необязательного предиката. Синтаксис шага определения расположения выглядит следующим образом: имя оси, двойное двоеточие, элемент проверки узла, нуль либо дополнительные предикаты, каждый из которых заключен в квадратные скобки. Базовая форма синтаксиса имеет следующий вид:

ось::узел[предикат]

- ось определяет древовидную связь между узлом контекста и узлами, которые выбираются шагом определения расположения. Другими словами, ось указывает общее направление, в котором выполняется шаг определения расположения по отношению к узлу контекста. В шаге определения расположения ось является необязательным элементом. Если не указать ось, по умолчанию ей присваивается значение child::.Кроме того, некоторые оси имеют ярлыки, например символ @ является ярлыком для оси атрибутов. Список осей представлен в таблице 10.

Таблица 10. Описание осей

Оси	Описание
ancestor::	Предки узла контекста. К предкам узла контекста относятся родительский узел узла контекста, родитель родителя и т. д.; таким образом, ось ancestor:: всегда включает в себя корневой узел, если только контекстный узел сам не является корневым узлом.
ancestor-or-self::	Узел контекста и его предки. Ось ancestor-or-self:: всегда включает корневой узел.
attribute::	Атрибуты контекстного узла. Ось будет пуста, если узел контекста не элемент.
child::	Дочерние элементы узла контекста. Дочерним является любой узел, расположенный на дереве непосредственно под узлом контекста. Однако ни узлы атрибутов, ни узлы пространства имен не рассматриваются в качестве дочерей узла контекста.
descendant::	Потомки контекстного узла. Потомком является дочерний объект или дочерний объект дочернего объекта и т. д.; таким образом, ось descendant:: никогда не содержит узлов атрибутов или пространства имен.
descendant-or-self::	Узел контекста и его потомки.
following::	Все узлы, расположенные на дереве после узла контекста, за исключением потомков, узлов атрибутов и узлов пространства имен.
following-sibling::	Все следующие элементы узла контекста с общим родителем. На оси following-sibling:: указываются лишь те дочерние объекты родительского узла, которые отображаются на дереве после узла контекста. На этой оси не указываются все другие дочерние объекты, расположенные перед узлом контекста. Если узел контекста является узлом атрибута или узлом пространства имен, ось following-sibling:: пуста.

namespace::	Узлы пространства имен узла контекста. На каждое пространство имен, расположенное в области узла контекста, приходится по одному узлу пространства имен. Ось будет пуста, если узел контекста не элемент.
parent::	Родитель узла контекста, если таковой имеется. Родителем является узел, расположенный на дереве непосредственно над узлом контекста.
preceding::	Все узлы, расположенные на дереве перед узлом контекста, за исключением предков, узлов атрибутов и узлов пространства имен. Чтобы лучше понять, что такое предшествующая ось, нужно представить себе все узлы, содержимое которых во всей своей полноте размещается до начала узла контекста.
preceding-sibling::	Все предшествующие элементы узла контекста с общим родителем. На оси preceding-sibling:: указываются лишь те потомки родительского узла, которые отображаются на дереве до узла контекста. На этой оси не указываются все другие дочерние объекты, расположенные после узла контекста. Если узел контекста является узлом атрибута или узлом пространства имен, ось preceding-sibling:: пуста.
self::	Только сам контекстный узел

- Узел определяет тип узла либо развернутое имя узлов, которые изначально выбираются шагом определения расположения. Проверка узлов является единственной необходимой частью шага определения расположения XPath. С учетом этого обстоятельства понимание ее является важнейшим условием успешного применения выражений XPath. Существует три основных типа проверки узлов: проверка имени, в которой используется развернутое имя и связь этого имени с заданной осью для указания узлов, которые необходимо выделить; проверка типа узла, при которой узлы выбираются строго по типам узлов; проверка целевой инструкции по обработке, при которой выделяются только те инструкции по обработке узлов, которые отвечают указанному типу.

- Предикат использует выражение XPath (условие, которое должно быть выполнено) для дальнейшего уточнения набора узлов, выбранного шагом определения расположения. Предикат представляет собой фильтр, который определяет критерий выбора для дальнейшего уточнения списка узлов-кандидатов .Предикат является необязательным элементом. Если предикат не указан, в шаге определения расположения нет квадратных скобок ([]). Собственно процесс фильтрации включает в себя последовательное вычисление предиката для каждого узла в наборе. Каждый раз, когда предикат вычисляется для узла, происходит следующее:
  Контекстным узлом является узел, для которого предикат вычисляется в данный момент. Размер контекста представляет собой число узлов в наборе, для которого вычисляется предикат. Позиция контекста представляет собой положение контекстного узла в наборе узлов. Этот последний контекст, положения контекстного узла в наборе узлов, является относительным и зависит от направления, в котором ось, заданная в шаге определения расположения данных, обходит дерево документа. Обычно ось обходит дерево в прямом или обратном направлении.

Набор узлов, выбираемый шагом определения расположения, является результатом создания исходного набора узлов, основанного на связи между осью и проверкой узла, и последующей фильтрации исходного набора по каждому включенному предикату. Исходный набор состоит из узлов, для которых выполняются два следующих условия. Узлы имеют связь с узлом контекста, заданным осью. Узлы имеют тип и развернутое имя, заданное проверкой узла. Затем выражение XPath использует первый предикат в шаге определения расположения, выполняя фильтрацию исходного набора узлов для создания нового набора узлов. Далее выражение XPath использует второй предикат для фильтрации набора узлов, полученного с помощью первого предиката. Процесс фильтрации повторяется до тех пор, пока выражение XPath не оценит все предикаты. Набор узлов, полученный в результате применения всех предикатов, является набором узлов, выбранным шагом определения расположения.

4.1.3. Контекст XPath-выражений

Оценка выражения XPath зависит от контекста, к которому обращается выражение. Контекст состоит из узла, по которому оценивается выражение, а также из связанной с ним среды, включающей следующие компоненты:

- Положение узла контекста в порядке документа относительно одноуровневых элементов.
- Размер контекста, т. е. число одноуровневых элементов узла контекста плюс один.
- Привязки переменных, с которыми разрешаются ссылки на переменные.
- Библиотека функций.
- Объявления пространств имен в области выражения.

Для того, чтобы лучше понять концепцию контекста, представьте дерево, содержащее узлы. Запрос всех узлов X из корня дерева возвратит один набор результатов, в то время, как запрос этих узлов из ветви дерева возвратит другой набор результатов. Таким образом, результат выражения зависит от контекста, к которому оно обращается при выполнении.

Выражения XPath могут сопоставить специальные шаблоны в одном конкретном контексте, затем вернуть результаты и выполнить дополнительные обращения к контексту возвращенных узлов. Это обеспечивает выражениям XPath исключительную гибкость при поиске по дереву документа.

4.1.4. Базовые выражения XPath

Ниже приведены базовые типы выражений XPath:

- Текущий контекст;
- Корень документа;
- Корневой элемент;
- Рекурсивный спуск;
- Конкретный элемент;

Текущий контекст - выражение с префиксом в виде точки и косой черты явным образом использует в качестве контекста текущий контекст. Например, следующее выражение ссылается на все элементы <author> внутри текущего контекста:

./author

Это выражение эквивалентно следующему:

Author

Корень документа - Выражение с префиксом в виде косой черты (/) использует в качестве контекста корень дерева документа. Например, следующее выражение ссылается на все элементы <bookstore> в корне этого документа:

/bookstore

Корневой элемент - выражение, использующее косую черту и звездочку (/*), использует в качестве контекста корневой элемент. Например, следующее выражение находит корневой элемент документа:

/*

Рекурсивный спуск - выражение, использующее двойную косую черту (//), указывает на поиск, который может включать ноль или более уровней иерархии. Если этот оператор отображается в начале шаблона, то контекст является относительным по отношению к корню документа. Например, следующее выражение ссылается на все элементы <author> внутри в любом месте внутри текущего документа:

//author

Префикс .// указывает, что контекст начинается на уровне иерархии, указанном в текущем контексте.

Конкретные элементы - выражение, которое начинается с имени элемента, ссылается на запрос конкретного элемента, который начинается от текущего узла контекста. Например, следующее выражение ссылается на элемент <background.jpg> внутри элемента <images> в текущего узле контекста:

images/background.jpg

Следующее выражение ссылается на коллекцию элементов <book> внутри элементов <bookstore> в текущего узле контекста:

bookstore/book

Представленное далее выражение ссылается на все элементы <first.name> внутри текущего узла контекста:

first.name

4.2. Язык преобразований XSLT

XSLT (eXtensible Stylesheet Language Transformations) - это декларативное описание преобразования (трансформации) любого XML-документа. Спецификация XSLT входит в состав XSL и является рекомендацией W3C.

Существует три основных способа преобразования XML-документов с помощью XSLT в другие форматы, например, в HTML:

- XML-документ и связанная с ним таблица стилей отправляются клиенту (веб-браузеру), который преобразует документ как указано в таблице стилей, и после этого представляет результат пользователю.
- Сервер применяет таблицу стилей XSLT к XML-документу и преобразует его в другой формат (обычно, в HTML). После этого результат отправляется клиенту (веб-браузеру).
- Какая-то программа преобразует оригинальный XML-документ в другой формат (обычно, в HTML), затем результат помещается на сервер. И сервер, и клиент имеет дело с уже преобразованным документом (рис.4.2.).

К разным документам можно применять разные таблицы стилей – и каждый раз получать различный результат.

Для обращения к элементам XML-документа XSLT использует XPath. Для объявления таблицы XSLT используется корневой элемент <xsl:stylesheet> (возможен вариант <xsl:transform>, варианты равнозначны):

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

Xml-файл, содержащий таблицу XSLT, состоит из набора шаблонов. Шаблон – это набор инструкций для преобразования входного XML-документа. Для создания шаблонов используется элемент <xsl:template>:

```
<xsl:template match = “XPath-выражение”>
        Тело шаблона
</xsl:template>
```

Структура элемента <xsl:template>:

```
<xsl:template
        match = pattern
```

```
        name = qname
        priority = number
        priority = qname>
            Тело шаблона
</xsl:template>
```

Правило шаблона задается элементом <xsl:template>. Атрибут match соответствует пути XPath, который идентифицирует исходные узел или узлы, к которым это правило применяется. Если элемент <xsl:template> не имеет атрибута name, атрибут match обязателен. Атрибуты priority и mode определяют последовательность и обязательность выполнения шаблонов. Содержимое элемента <xsl:template> является шаблоном, который обрабатывается если данное правило шаблона задействовано.

Для получения значений элемента XML и вывода его, например, на экран, используется элемент <xsl:value-of>:

```
<xsl:template match="employee">
        <xsl:value-of select="name" />
  </xsl:template>
```

Элемент <xsl:apply-templates> дает указание преобразователю сравнивать каждый дочерний элемент исходного элемента (который соответствует данному шаблону) с другими шаблонами в таблице стилей и, если соответствие обнаружено, выводить шаблон для соответствующего узла.

В отсутствие атрибута select инструкция <xsl:apply-templates> обрабатывает все дочерние узлы текущего узла, включая узлы текста:

```
<xsl:template match="chapter">
        <fo:block>
            <xsl:apply-templates/>
        </fo:block>
</xsl:template>
```

Чтобы обрабатывать не все дочерние узлы, а лишь узлы, отобранные по некому выражению, может использоваться атрибут select. Значением атрибута select является XPath-выражение. После обработки этого выражения должен получиться набор узлов.

Шаблоны можно вызывать по имени. Именованный шаблон задается элементом <xsl:template> с атрибутом name. Элемент <xsl:call-template> вызывает шаблон по имени name, идентифицирующий шаблон, который должен быть вызван.

Синтаксис именованного шаблона:

```
<xsl:template name = "MyTemplate">
        Тело шаблона
</xsl:template>
```

Синтаксис вызова именованного шаблона:

```
<xsl:call-template name = "MyTemplate">
        Тело шаблона
</xsl:template>
```

Создавать новые узлы в XML-документе можно с помощью конструкции <xsl:element>:

```
<xsl:element
        name = “elementName”
        namespace = “URI”
        use-attribute-sets = “nameList”>
            Содержание элемента
</xsl:element>
```

Элемент <xsl:element> позволяет создавать элемент с вычисляемым названием. Единственным обязательным атрибутом данной конструкции является атрибут name.

Новые атрибуты создаются с помощью конструкции <xsl:attribute>:

```
<xsl:attribute
        name = “attrName”
        namespace = “URI”>
            Содержание атрибута
</xsl:attribute>
```

Чтобы к конечным элементам добавить атрибуты, можно использовать элемент <xsl:attribute> независимо от того, созданы ли первые фиксированными конечными элементами или такими инструкциями, как <xsl:element>.

Шаблон также может содержать текстовые узлы. Каждый текстовый узел в шаблоне, оставшийся после удаления пробельных символов, создаст в конечном дереве текстовый узел с тем же самым строковым значением. Смежные текстовые узлы в конечном дереве автоматически объединяются. Синтаксис создания текстового узла:

```
<xsl:text
        disable-output-escaping = "yes" | "no">
        <!--Содержание текстового узла: #PCDATA -->
</xsl:text>
```

Для создания в конечном дереве узла комментариев используется элемент <xsl:comment>. Содержимое элемента <xsl:comment> является шаблоном для строкового значения данного узла комментария:

```
<xsl:comment>
      <!—Текст комментария -->
</xsl:comment>
```

Для создания узла инструкции обработки (процессинговых инструкций) используется элемент <xsl:processing-instruction>. Содержимое элемента <xsl:processing-instruction> является шаблоном для строкового значения узла инструкции обработки. Элемент <xsl:processing-instruction> имеет обязательный атрибут name, который определяет название данного узла инструкции обработки:

```
<xsl:processing-instruction
        name = “name”>
            <!-- Содержание инструкции -->
</xsl:processing-instruction>
```

Ниже приведен пример XSLT-преобразования XML-документа. Исходный XML-документ:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="emp1.xsl"?>

<employees xmlns:au="http://www.demotest.com">
 <employee>
        <name>Adam Stein</name>
          <salary>23500</salary>
          <jobtitle>Programmer</jobtitle>
          <region>Redmond</region>
 </employee>
 <employee>
         <name>Susan Tjarnberg</name>
         <salary>51000</salary>
         <jobtitle>Tester</jobtitle>
          <region>Minneapolis</region>
```

```
</employee>
<employee>
        <name>Catherine Turner</name>
        <salary>45000</salary>
       <jobtitle>System Architect</jobtitle>
        <region>Dallas</region>
</employee>
<employee>
        <name>Wendy Vasse</name>
        <salary>72000</salary>
        <jobtitle>Project Manager</jobtitle>
        <region>Washington D.C.</region>
</employee>
<employee>
         <name>Paula Thurman</name>
        <au:salary>34500</au:salary>
         <jobtitle>Programmer</jobtitle>
        <region>Sydney</region>
        </employee>
<employee>
       <name>Richard Marshall</name>
        <au:salary>30000</au:salary>
       <jobtitle>Programmer</jobtitle>
        <region>Canberra</region>
</employee>
</employees>
```

Таблица преобразований XSLT:

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

<xsl:template match="/">
  <HTML>
       <BODY>
            <H1>Salaries Report</H1>
            <xsl:apply-templates/>
  </BODY>
  </HTML>

 </xsl:template>
       <xsl:template match="employees">
```

```
        <xsl:apply-templates select="employee[salary]"/>
        <xsl:apply-templates select="employee[au:salary]"/>
 </xsl:template>
<xsl:template match="employee[salary]">
          <p><xsl:value-of select="name"/>, <xsl:value-of select="jobtitle"/>,
       <xsl:value-of select="region"/>, <xsl:value-of select="salary"/> USD
(US dollars)</p>
 </xsl:template>
<xsl:template match="employee[au:salary]">
  <p><xsl:value-of select="name"/>,
<xsl:value-of select="jobtitle"/>, <xsl:value-of select="region"/>,
<xsl:value-of select="au:salary"/> AUD (Australian dollars)</p>
 </xsl:template>
</xsl:stylesheet>
```

Код, получившийся в результате преобразования:

```
<HTML xmlns:au="http://www.demotest.com">
        <BODY>
                <H1>Salaries Report</H1>
                <p>Adam Stein, Programmer, Redmond,
                        23500 USD (US dollars)</p>
                <p>Susan Tjarnberg, Tester, Minneapolis,
                        51000 USD (US dollars)</p>
                <p>Catherine Turner, System Architect, Dallas,
                        45000 USD (US dollars)</p>
                <p>Wendy Vasse, Project Manager, Washington D.C.,
                        72000 USD (US dollars)</p>
                <p>Paula Thurman, Programmer, Sydney,
                        34500 AUD (Australian dollars)</p>
                <p>Richard Marshall, Programmer, Canberra,
                        30000 AUD (Australian dollars)</p>
        </BODY>
</HTML>
```

Отображение результата в браузере (Рис.1):

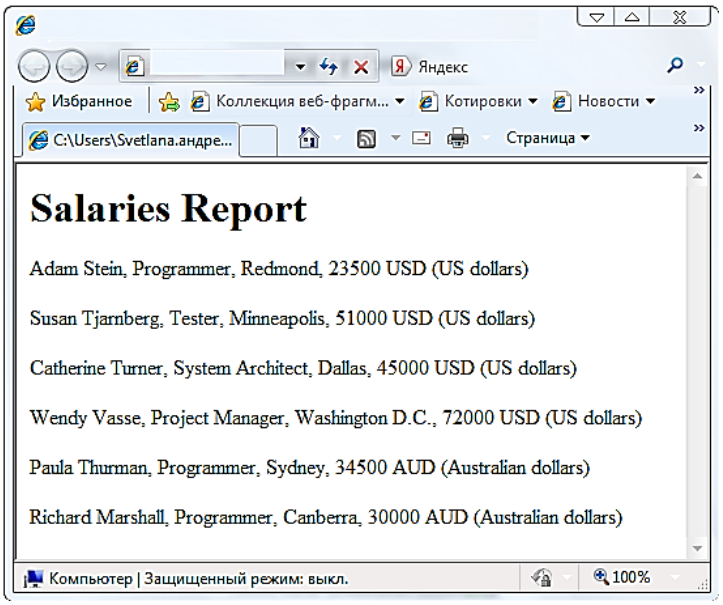

Рис.1. Результат XSLT-преобразования XML-документа в HTML

Контрольные вопросы

1. Для чего разработан язык XPath?
2. Как прописываются пути в XPath?
3. Что определяет ось выборки XPath?
4. С какой целью в XPath используются предикаты?
5. Чем определяется контекст выражения в XPath?
6. Каково основное назначение технологии XSLT?
7. Как строятся шаблоны преобразований в XSLT?
8. Какова структура шаблона XSLT?
9. Как с помощью языка XSLT создаются новые структурные единицы документа (элементы, атрибуты и т.п.)?
10. Каков алгоритм преобразования XML-документа с помощью языка XSLT?

Литература

1. Bonqers F. XSLT 2.0 und XPath 2.0, Galileo Press Gmbh, 2008 – 1151 c.
2. Fawcett J., Ayers D., Quin L. Beginning XML, 5th Edition. Wrox, 2012 – 864 c.
3. Holzner S. XML: A Beginner's Guide: Go Beyond the Basics with Ajax, XHTML, XPath 2.0, XSLT 2.0 and XQuery. McGraw-Hill Osborne Media, 2008 – 456 c.
4. Kay M. XSLT 2.0 and XPath 2.0 Programmer's Reference. Wrox, 2008 – 1368 c.
5. Melton J., Buxton S. Querying XML, : XQuery, XPath, and SQL/XML in context. Morgan Kaufmann, 2006 – 848 c.
6. Schema // URL: http://www.w3.org/standards/xml/schema.html
7. Sebastian J. The Art of XSD - SQL Server XML schemas. Red gate books, 2009 – 464 c.
8. Sikos L. Web Standards: Mastering HTML5, CSS3, and XML. Apress, 2011 – 524 c.
9. Transformation // URL: http://www.w3.org/standards/xml/transformation.html
10. Walmsley P. Definitive XML Schema, 2nd Edition. Prentice Hall, 2012 – 768 c.
11. XLXP,XSLT,XPATH,XFORMS & XQuery Interview Questions You'll Most Likely Be Asked / Vibrant Publisher, CreateSpace Independent Publishing Platform, 2012 – 100 c.
12. XML Essentials // URL: http://www.w3.org/standards/xml/core
13. XML Path Language (XPath) 2.0 (Second Edition) // URL: http://www.w3.org/TR/2010/REC-xpath20-20101214/
14. XML Technology // URL: http://www.w3.org/standards/xml/
15. XPath Current Status // URL: http://www.w3.org/standards/techs/xpath#w3c_all
16. XSL Transformations (XSLT) Version 2.0 // URL: http://www.w3.org/TR/2007/REC-xslt20-20070123/
17. XSLT Current Status // URL: http://www.w3.org/standards/techs/xslt#w3c_all

18.Голощапов А. Microsoft Visual Studio 2010. СПб.: БХВ-Петербург, 2011 – 544 с.

19.Красиков И. XML. Базовый курс. М.: Вильямс, 2009 – 1344 с.

20.Кригель А., Трухнов Б. SQL. Библия пользователя. М.: Вильямс, 2010 – 752 с.

21.Макки А. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов. М.: Вильямс, 2010 – 416 с.

22.Мангано С. XSLT. Сборник рецептов. СПб.: БХВ-Петербург, 2008 – 864 с.

23.Рендольф Н., Гарднер Д., Минутилло М., Андерсон К. Visual Studio 2010 для профессионалов. М.: Диалектика, 2011 – 1184 с.

24.Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. СПб.: Питер, 2012 – 928 с.

25.Рэй Э. Изучаем XML. М.: Символ-Плюс, 2001 – 546 с.

26.Смирнов С.Н. XML и JDBC. Практическое введение. М.:Гелиос АРВ, 2010 – 188 с.

27.Тидуэлл Д. XSLT. М.: Символ-Плюс, 2010 – 960 с.

28.Троелсен Э. Язык программирования C# 2010 и платформа .NET 4. М.: Вильямс, 2011 – 1392 с.

29.Уильямс К., Грэй Д, Джоши Б., Палермо М., Нортон Ф., Нормен Ф., Сингх Д., Слэйтер Д., Дальви Д. XML.NET. М:Лори, 2012 – 642 с.

30.Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. СПб.: Питер, 2012 – 784 с.

31.Цвалина К., Абраис Б. Инфраструктура программных проектов. Соглашения, идиомы и шаблоны для многократно используемых библиотек .NET. М.: Вильямс, 2011 – 416 с.

32.Шилдт Г. C# 4.0 полное руководство. М.: Вильямс, 2011 – 1056 с.

33.Эллоит Р., Минс С. XML. Справочник. М.: Символ-Плюс, 2001 – 576 с.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Кафедра **Программных систем** входит в состав нового факультета **Инфокоммуникационные технологии**, созданного решением Ученого совета университета 17 декабря 2010 г. по предложению инициативной группы сотрудников, имеющих большой опыт в реализации инфокоммуникационных проектов федерального и регионального значения.

На кафедре ведется подготовка бакалавров и магистров по направлению **210700 «Инфокоммуникационные технологии и системы связи»:**
210700.62.10 – ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ (Бакалавр)
210700.68.05 – ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ В ИНФОКОММУНИКАЦИЯХ (Магистр)

Выпускники кафедры получают фундаментальную подготовку по: математике, физике, электронике, моделированию и проектированию инфокоммуникационных систем (ИКС), информатике и программированию, теории связи и теории информации.

В рамках профессионального цикла изучаются дисциплины: архитектура ИКС, технологии программирования, ИКС в Интернете, сетевые технологии, администрирование сетей Windows и UNIX, создание программного обеспечения ИКС, Web программирование, создание клиент-серверных приложений.

Область профессиональной деятельности бакалавров и магистров включает:
- сервисно-эксплуационная в сфере современных ИКС;
- расчетно-проектная при создании и поддержке сетевых услуг и сервисов;
- экспериментально-исследовательская;
- организационно-управленческая – в сфере информационного менеджмента ИКС.

Знания выпускников востребованы:
- в технических и программных системах;
- в системах и устройствах звукового вещания, электроакустики, речевой, и мультимедийной информатики;

- в средствах и методах защиты информации;
- в методах проектирования и моделирования сложных систем;
- в вопросах передачи и распределения информации в телекоммуникационных системах и сетях;
- в методах управления телекоммуникационными сетями и системами;
- в вопросах создания программного обеспечения ИКС.

Выпускники кафедры Программных систем обладают компетенциями:
- проектировщика и разработчика структур ИКС;
- специалиста по моделированию процессов сложных систем;
- разработчика алгоритмов решения задач ИКС;
- специалиста по безопасности жизнедеятельности ИКС;
- разработчика сетевых услуг и сервисов в ИКС;
- администратора сетей: UNIX и Windows;
- разработчика клиентских и клиент-серверных приложений;
- разработчика Web – приложений;
- специалиста по информационному менеджменту;
- менеджера проектов планирования развития ИКС.

Трудоустройство выпускников:
1. ОАО «Петербургская телефонная сеть»;
2. АО «ЛЕНГИПРОТРАНС»;
3. Акционерный коммерческий Сберегательный банк Российской Федерации;
4. ОАО «РИВЦ-Пулково»;
5. СПБ ГУП «Петербургский метрополитен»;
6. ООО «СоюзБалтКомплект»;
7. ООО «ОТИС Лифт»;
8. ОАО «Новые Информационные Технологии в Авиации»;
9. ООО «Т-Системс СиАйЭс» и др.

Кафедра сегодня имеет в своем составе высококвалифицированный преподавательский состав, в том числе:
- 5 кандидатов технических наук, имеющих ученые звания профессора и доцента;
- 4 старших преподавателя;
- 6 штатных совместителей, в том числе кандидатов наук, профессиональных IT-специалистов;
- 15 Сертифицированных тренеров, имеющих Западные Сертификаты фирм: Microsoft, Oracle, Cisco, Novell.

Современная техническая база; лицензионное программное обеспечение; специализированные лаборатории, оснащенные необходимым оборудованием и ПО; качественная методическая поддержка образовательных программ; широкие Партнерские связи существенно влияют на конкурентные преимущества подготовки специалистов.

Авторитет специализаций кафедры в области компьютерных технологий подтверждается Сертификатами на право проведения обучения по методикам ведущих Западных фирм - поставщиков аппаратного и программного обеспечения.

Заслуженной популярностью пользуются специализации кафедры ПС по подготовке и переподготовке профессиональных компьютерных специалистов с выдачей **Государственного Диплома** о профессиональной переподготовке по направлениям:

"Информационные технологии (инженер-программист)" и **"Системный инженер"**, а также Диплома о дополнительном (к высшему) образованию с присвоением квалификации: **"Разработчик профессионально-ориентированных компьютерных технологий "**. В рамках этих специализаций высокопрофессиональные преподаватели готовят компетентных компьютерных специалистов по современным в России и за рубежом операционным системам, базам данных и языкам программирования ведущих фирм: Microsoft, Cisco, IBM, Intel, Oracle, Novell и др

Профессионализм, компетентность, опыт, и качество программ подготовки и переподготовки IT-специалистов на кафедре ПС неоднократно были удостоены **высокими наградами «Компьютерная Элита» в номинации лучший учебный центр России**.

Партнеры:

1. **Microso**ft Certified Learning Solutions;
2. **Novell** Authorized Education Center;
3. **Cisco** Networking Academy;
4. **Oracle** Academy;
5. **Sun Java** Academy и др;
6. **Prometric**;
7. **VUE**.

Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.

С.В. Одиночкина

Основы технологий XML

УЧЕБНОЕ ПОСОБИЕ

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж 100 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики
и оптики

197101, Санкт-Петербург, Кронверкский пр., 49