Лекция 7

XML и .NET

- Краткое описание стандарта XML
- Об использовании XML в .NET
- Некоторые полезные классы из System.Xml

Некоторые факты о XML

- XML = eXtensible Markup Language
- Разработан соответствующей рабочей группой концерна W3C в 1996 г., стандартизован в 1998 г.
- Очень простое описание - около 30 стр.
- Современная версия XML - 1.0

Предыстория XML: SGML

- 1980-е - SGML (Standard Generalized Markup Language)
  - **разработан для МО США; задача - снизить расходы на передачу документации**
  - **четкая иерархическая структурированность информации;**
  - **расширяемость стандарта**
  - **отделение информации от представления (использование DTD - Document Type Definition)**
  - **слишком сложен для реализации в Web (Sounds Good, Maybe Later)**

Предыстория XML: HTML

- 1991 - HTML (HyperText Markup Language)
  - **фиксированное подмножество SGML, ориентированное на Web-страницы**
  - **чрезвычайно распространен - существует около 100 млрд. HTML-страниц!**
  - **трудности форматирования текста и невозможность отделить представление от данных (CSS решает эту проблему только частично)**
  - **недостаточная строгость стандарта и, как следствие, проблема структурирования текста**

Зачем нам еще один стандарт?

- Затем, что необходимо компактное, дешевое, простое и быстрое средство, аналогичное HTML, которое могло бы еще и расширяться, как SGML.
- Еще было бы здорово иметь единый формат для передачи данных между приложениями
- Еще хотелось бы создавать структурированные тексты с возможностью создания оглавлений, перекрестных ссылок и т.д.
- Еще нам нужно уметь отображать нестандартную информацию (математические формулы, ноты и т.п.)

Десять целей XML

- XML должен быть ориентирован на использование в Интернет
- XML должен быть пригоден для использования в большом количестве приложений
- XML должен быть совместим с SGML
- Создание программ, обрабатывающих XML, должно быть простым
- Число необязательных возможностей языка должно быть сведено к минимуму (в идеале - к нулю)

Десять целей XML (окончание)

- Документы XML должны быть достаточно просты и понятны человеку
- XML следует разработать в короткие сроки
- Спецификация XML должна быть четкой и краткой
- Создание XML-документов должно быть максимально простым
- Краткость команд разметки XML не имеет принципиального значения

Мифы об XML

- XML - это язык разметки (на самом деле - это метаязык для создания языков разметки)
- XML - это только для Web (на самом деле, многие компании переходят на него с SGML)
- XML - это подмножество SGML (раньше был, но теперь ввели схемы, пространства имен и т.д.)
- HTML - это подмножество XML (такое описание в принципе возможно, но бессмысленно; кроме того, есть вопрос следования стандарту HTML)
- XML - это еще один рекламный трюк (увы, нет)

Основы синтаксиса XML

- У каждого элемента должен быть открывающий и закрывающий тэг
- В документе должен быть ровно один корневой элемент
- Элементы могут быть вложены друг в друга, но не могут пересекаться (т.е. деревянная структура)
- Все названия элементов чувствительны к регистру
- Некоторые символы запрещены (надо использовать специальные последовательности)
- Значения атрибутов должны быть взяты в кавычки

Пример XML-документа

```
<?xml version="1.0"?>
<Joke author="Groucho Marx">
    <Setup>Outside of a dog, a book is
    man's best friend.</Setup>
    <Punchline>Inside of a dog, it's too
    dark to read.</Punchline>
</Joke>
```

Заголовок XML

- Необязательный, предназначен для синтаксического анализатора; должен идти с самого начала файла

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

- Может указывать кодировку файла (по умолчанию UTF-8 или UTF-16 (в отличие от HTML):

```
<?xml version="1.0" encoding="windows-1251"?>
```

- Может указывать атрибут "самодостаточности":

```
<?xml version="1.0" standalone="no"?>
```

- Заголовок чувствителен к регистру символов!

Структура тэгов XML

- Должен быть открывающий и закрывающий тэг:

```
<duration course=".NET">1.5</duration>
```

Здесь duration - это название тэга, course – это атрибут, а 1.5 – значение (содержимое элемента)

- Должны соблюдаться правила вложенности:

```
<P>Права <B>одного <I>человека</B> важнее</I>
прав коллектива - неправильно!!!
```

- Вместо этого правильно так:

```
<P>Права <B>одного <I>человека</I></B>
<I>важнее</I> прав коллектива
```

Корневой тэг

- Следующий пример некорректен:

```
<!-- WRONG! -->
<a> ... </a>
<b> ... </b>
```

- Вместо этого должно быть:

```
<root>
  <a> ... </a>
  <b> ... </b>
</root>
```

Особые символы XML

- Некоторые символы необходимо заменять на подстановочные строки:

```
<P>Было предложено соотношение A<B, но оно не подошло.</P>
```

- Должно быть:

```
<P>Было предложено соотношение A&lt;B, но оно не подошло.</P>
```

- Пять подстановочных строк:
  - **< > & ' "**
  - **< > & ' "**

Атрибуты

- Содержат уточняющую информацию об элементе

```
<insuranceClaim dateFiled="2001-04-17">
```

- Все значения атрибутов должны быть в одинарных или двойных кавычках! (в отличие от HTML)
- Где хранить данные - в атрибутах или в значениях?
  - Атрибуты - это свойства объекта (подобен прилагательному по отношению к существительному)
  - Атрибуты эффективнее (5 байт против 7 байт)
  - Атрибуты не могут иметь вложенные структуры, а элемент может
  - Если не можете решить, включайте данные в элемент

Пустые элементы и комментарии

- Иногда хочется создать элемент без содержания (например, горизонтальная линия в HTML, <HR>)
- Нужен закрывающий символ
- Упрощение синтаксиса:

```
<HR/> <IMG SRC="/images/dotNET.gif"/>
```

- Комментарии:

```
<!-- Некая полезная информация -->
```

Корректно сформированные документы

- Документы, удовлетворяющие описанным выше правилам, называются корректно сформированными (well-formed) документами

```
<configuration type="printer">
    <parm name="port">/usr/lpr</parm>
    <parm name="driver">/usr/drivers/HP5SIPS</parm>
    <parm name="option">sheet feeder</parm>
    <configured/>
    <online/>
</configuration>
```

Совместная обработка XML-файлов

```
<?xml version="1.0"?>
<favorite-joke author="Pate">
<one-liner>A duck walks into a bar and  says to the
bartender, "Gimme a shot of whisky and put it on my
bill."</one-liner>
</favorite-joke>

<?xml version="1.0"?>
<duck_joke>
<scene num="1">A duck walks into a bar and asks for the
whisky and a cucumber. Then it drinks whisky, eats his
cucumber and leaves.</scene>
<scene num="2">Then a cowboy sitting in the bar says to
another cowboy: "For the first time in my life I saw
someone finishing whiskey with a cucumber."</scene>
</duck_joke>
```

Проблема обработки XML-документов

- И первый, и второй примеры являются корректно сформированными XML-документами.
- Но можно ли создать программу, которая записывала бы эти и другие анекдоты в базу данных? По крайней мере, это будет сложно...
- Однако можно было бы разослать всем авторам некоторую заранее предопределенную структуру ожидаемого документа
- Такая структура для XML называется схемой документа. Стандартом на сегодня является DTD

Определение типа документа (DTD)

- "Документ называется действительным, если он имеет связанное с ним определение типа документа (схему) и соответствует ему"
- В XML 1.0 единственным типом схем является DTD – Document Type Definition
- DTD основано на упрощенном формате SGML и было создано для нужд EDI (Electronic Data Interchange)
- Синтаксис DTD существенно отличается от XML

Пример DTD

```
<!ELEMENT Joke       (Setup, Punchline)              >
<!ATTLIST Joke       author CDATA #REQUIRED
                     firstTold CDATA #IMPLIED        >
<!ELEMENT Setup      (#PCDATA)                       >
<!ELEMENT Punchline  (#PCDATA)                       >
```

```
<?xml version="1.0"?>
<!DOCTYPE Joke SYSTEM "Joke.dtd">
<Joke author="Groucho Marx">
    <Setup>Outside of a dog, a book is man's
    best friend</Setup>
    <Punchline>Inside of a dog, it's too dark
    to read.</Punchline>
</Joke>
```

Проблемы DTD

- DTD - строго иерархический формат, плохо подходящий для меняющихся документов
- У документа может быть только один тип, что не очень удобно для коммерческих приложений (пример с разными форматами адреса)
- Синтаксис DTD не похож на XML и плохо воспринимается человеком
- В DTD допустимо использование только текстового типа данных (нет числового типа или даты)

Предлагаемое решение: XML-схемы

- Схемы должны следовать синтаксису XML
- Стандарт схемы должен поддерживать распространенные типы данных (число, дата…)
- Схема XML должна быть открытой для подключения внешних источников
- Метод разрешения неоднозначности имен при соединении двух файлов - пространства имен:

```
<invoice xmlns="http://www.vasyapupkin.com/s.xdr">
<address xmlns="xml.org/schemas/us-address.xdr">
<name><invoice:title>Mr.</invoice:title>Pupkin</name>
```

Document Object Model

- Объектная модель документа открывает доступ к XML-документу как к древовидной структуре в памяти
- Позволяет работать с XML-документом как с обычным объектом в любом современном языке программирования
- Перед началом работы DOM требует загрузки всего документа в память (поэтому разрабатываются и альтернативные стандарты, например, SAX)
- Стандарт оставляет разработчикам большую свободу в интерпретации
- Microsoft предлагает MS XML DOM 3.0, интегрированную в IE 5, MS Office 2000 и MS Windows 2000

XLink & XPointer

- Иногда возникает ситуация, когда объект связан с другим объектом, но эта связь не является иерархической
- XLink позволяет создавать в XML-документе ссылки и задавать семантику его обработки (кроме того, поддерживает встроенные ссылки)
- XPointer позволяет ссылаться на фрагменты документов с гораздо большей степенью детализации, чем была доступна в HTML

XSL и XSLT

- XSL позволяет описывать внешний вид (форматирование) XML-документа
- Реально XSL состоит из двух документов: первый относится к преобразованию документов XML, а второй – к интерпретации форматирования
- Язык преобразования описан в стандарте XSLT (XSL Transformations)

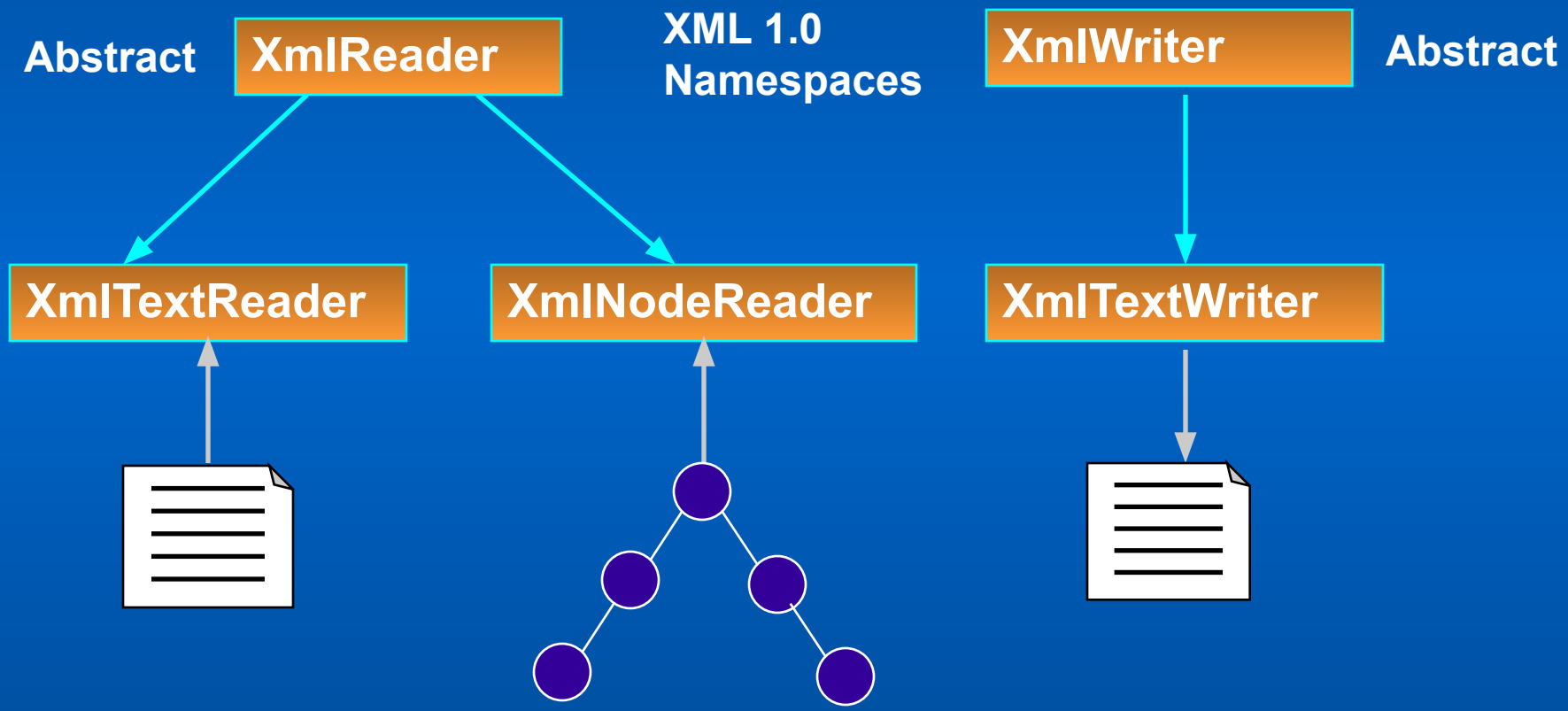
Разработка отраслевых схем

- Разрабатываются промышленными группами (автомобилестроение, химические и математические формулы и т.д.)
- Самая первая попытка создания такой отраслевой схемы – в 80-х гг. Air Transport Association (еще на SGML)
- Самый популярный пример сегодня – стандарт записи математических формул (MathML)

XML-компоненты в .NET

- **XmlReader & XmlWriter**
  - Могут читать/писать корректно сформированный XML
- **XmlDocument**
  - На базе модели W3C DOM с поддержкой XPath SelectNodes() и SelectSingleNode()
- **XPathNavigator**
  - Предоставляет модель XPath поверх любых данных
  - XPathDocument (XML-данные, оптимизированные для X/Path)
- **XSLTransform**
  - Предоставляет трансформации над XPath
- **XSD Compliance**
  - XmlSchema - Object Model (SOM)
  - XmlValidatingReader

XmlReader и XmlWriter



Описание XmlReader и XmlWriter

- XmlReader
  - Основан на модели Pull; основная схема применения:
    - While (reader.Read())
      { /* обработка полученных узлов */}
  - Предоставляет последовательный (forward-only) курсор над любыми XML-данными
  - Реализован в XmlTextReader, XmlNodeReader
- XmlWriter
  - Генерирует валидный XML
  - Помогает обрабатывать пространства имен
  - Реализован в XmlTextWriter

Пример использования XmlTextReader

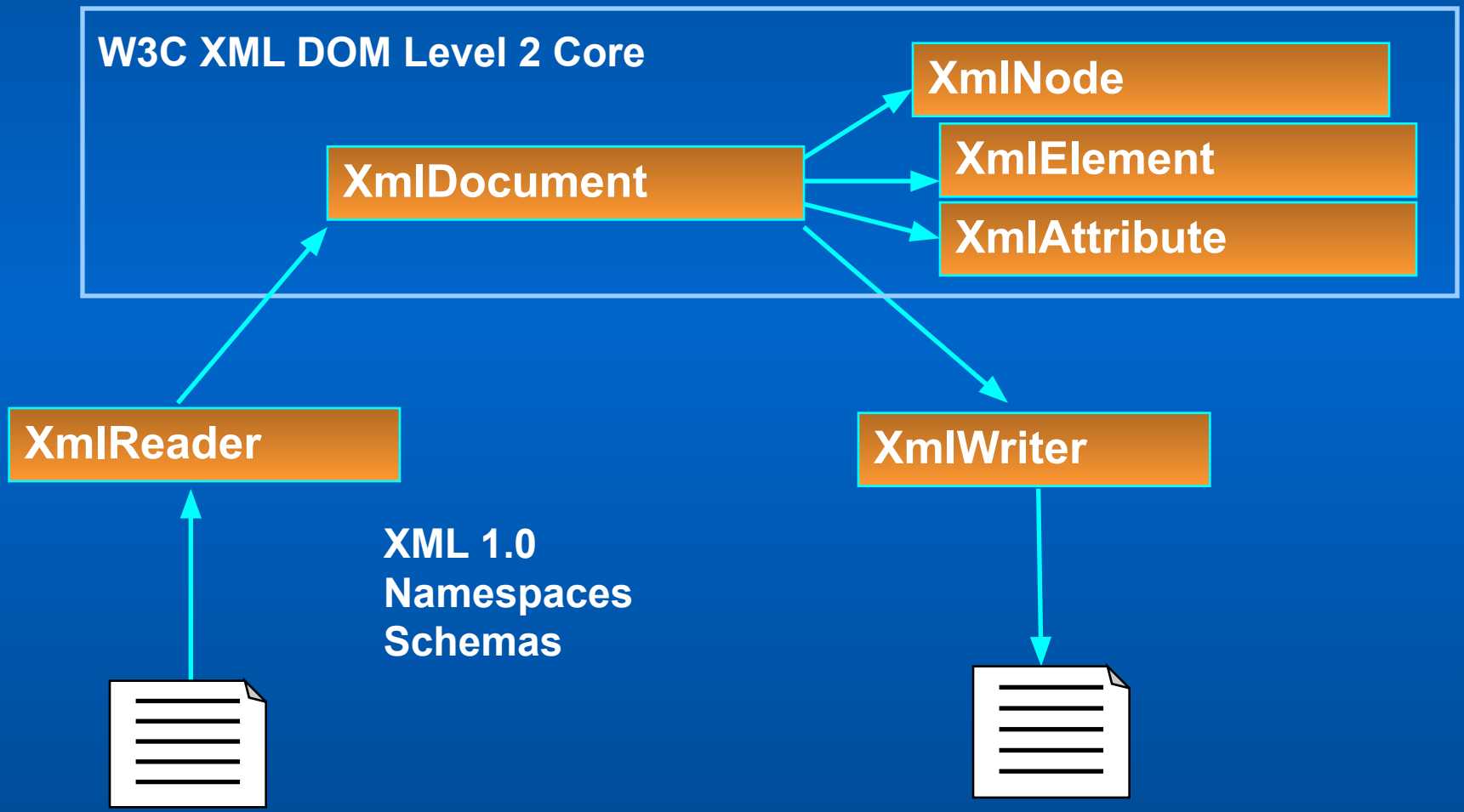
```
XmlTextReader xr = new XmlTextReader("MyFile.Xml");
while (xr.Read()) {
        switch (xr.NodeType) {
                case XmlNodeType.Document:
                        Console.Write("<?xml version='1.0'?>");
                        break;
                case XmlNodeType.Element:
                        Console.Write("<" + xr.Name+">");
                        break;
                case XmlNodeType.SignificantWhitespace:
                case XmlNodeType.Text:
                        Console.Write(xr.Value);
                        break;
                case XmlNodeType.Comment:
                        Console.Write("<!--" + xr.Value + "-->");
                        break;
                case XmlNodeType.ProcessingInstruction:
                        Console.Write("<?" + xr.Name + " " + xr.Value + "?>");
                         break;
                case XmlNodeType.EndElement:
                        Console.Writeline("</" + xr.Name + ">");
                }
        }
```

Пример использования XmlWriter

```
public void WriteDocument(XmlWriter writer) {
  writer.WriteStartDocument();
  writer.WriteComment(“sample person document");
  writer.WriteProcessingInstruction("hack", "on person");
  writer.WriteStartElement("p", "person", "urn:person");
  writer.WriteStartElement("name", "");
  writer.WriteString("joebob");
  writer.WriteEndElement();
  writer.WriteElementInt16("age", "", 28);
  writer.WriteEndElement();
  writer.WriteEndDocument();
}
```

```
<?xml version="1.0"?>
<!--sample person document-->
<?hack on person?>
<p:person xmlns:p="urn:person">
  <name>joebob</name>
  <age unit="year">28</age>
</p:person>
```

Архитектура XmlDocument



XmlDocument

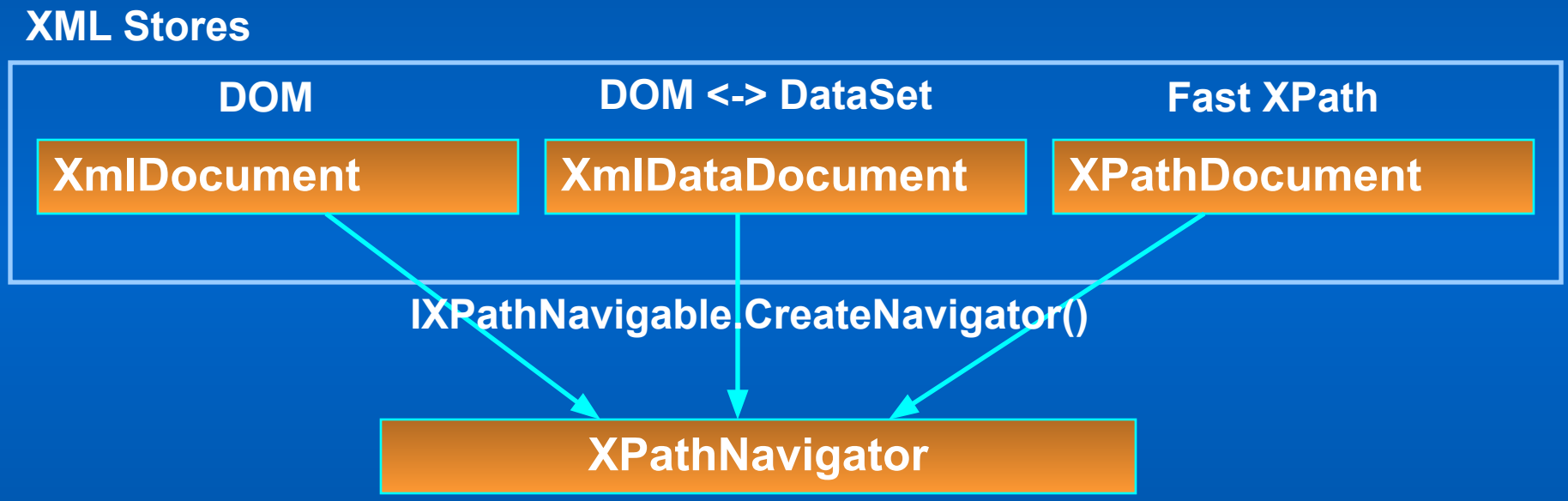
- Поддерживает рекомендации W3C DOM Core Level 1 и Core Level 2 (пространства имен)
- Предоставляет API для редактирования XML
- MS-расширения – Load(), Save() и обработка событий
- Свойство InnerXml (аналог innerHTML)

```
node.InnerXml =
  "<Person><Name>John</Name><Email>
  john@microsoft.com</Email></Person>"
```

Пример использования XmlDocument

```
using System; using System.Xml;

public class GenerateDocument {
  public static void Main(String[] args) {
    // instantiate the document
    XmlDocument doc = new XmlDocument(); XmlNode node;

    // create a comment/pi and append
    node = doc.CreateComment("sample person document");
    doc.AppendChild(node);
    node = doc.CreateProcessingInstruction("hack", "on person");
    doc.AppendChild(node);

    // create the person element within the 'urn:person' namespace
    node = doc.CreateElement("p", "person", "urn:person");
    doc.AppendChild(node); node = doc.CreateElement("name");
    node.InnerText = "joebob"; doc.DocumentElement.AppendChild(node);
    node = doc.CreateElement("age"); node.InnerText = "28";
    doc.DocumentElement.AppendChild(node);

    // serialize the document to the console
    XmlTextWriter tw = new XmlTextWriter(Console.Out);
    tw.Formatting = Formatting.Indented; doc.Save(tw);  } }
```

Архитектура XPath



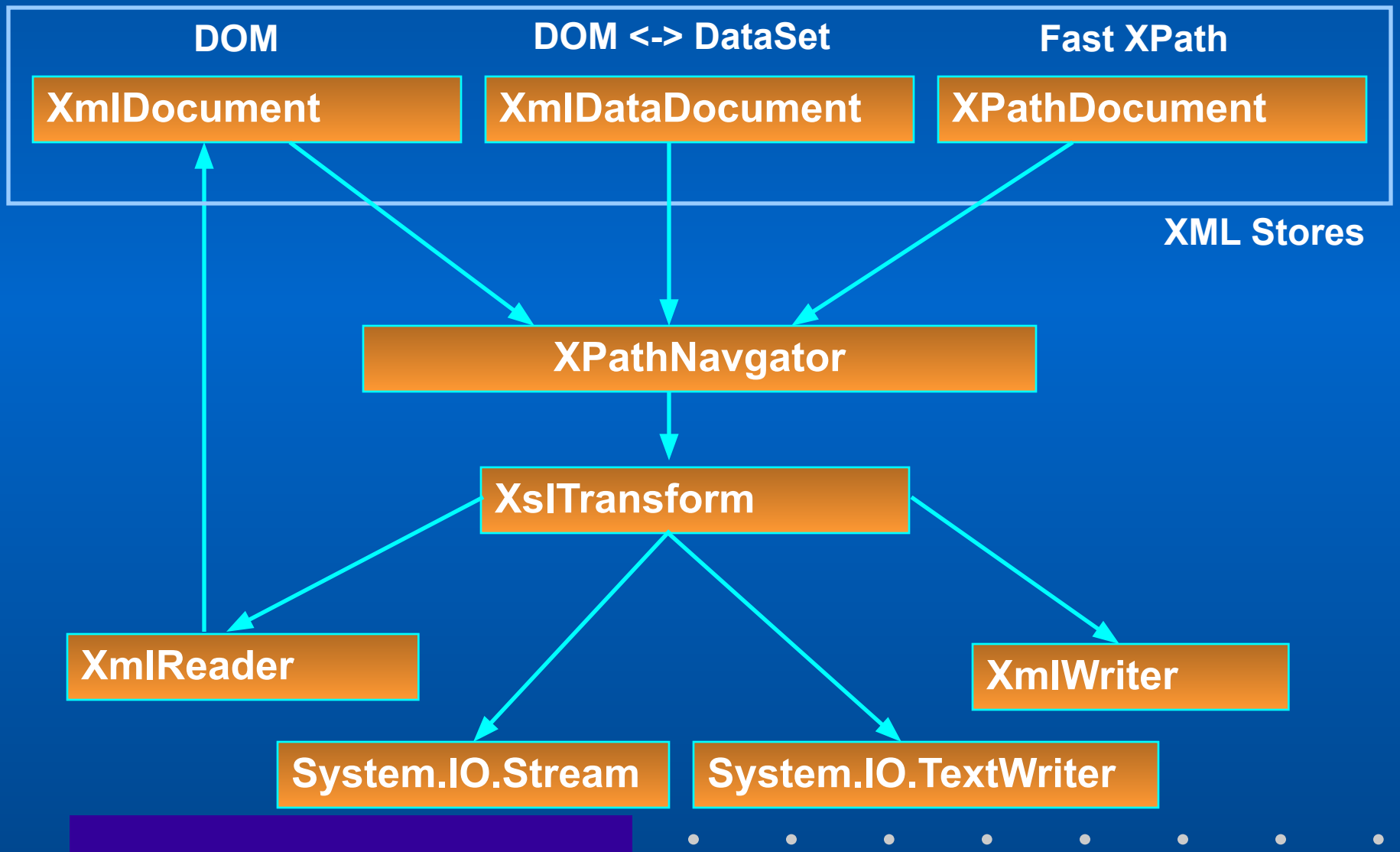
XPathNavigator

- Предоставляет возможность использования модели XPath над любым типом данных
- Доступ к данным XML в стиле обычного курсора
- Предоставляет функциональность XPath:
  - void Select (String xpath);
  - Object Evaluate (String xpath);
- XPathNodeIterator – итератор для прохождения всех выбранных узлов

Пример использования XPath

```
// Создаем XPathDocument
XPathDocument myXPathDoc = new XPathDocument("books.xml");
// Получаем XPathNavigator
XPathNavigator myNav =
   ((IXPathNavigable)myXPathDoc).CreateNavigator();

// Печатаем цену каждой книги (полный обход дерева)
XPathNodeIterator iter =
   myNav.Select("descendant::book/price");
while(iter.MoveNext())
   Console.WriteLine(iter.Current.Value);

// Печатаем общую сумму цен всех книг (без обхода дерева)
Console.WriteLine ("Total Price: {0} ",
   myNav.Evaluate ("sum(descendant::book/price)"));
```

Архитектура XslTransform



Использование XslTransform

- XPathDocument дает нам оптимизированный формат для чтения данных с использованием XPath and XSL/T:

```
XPathDocument doc = new XPathDocument("XmlDoc.xml");
XslTransform t = new XslTransform();
t.Load( Stylesheet );
XmlTextWriter writer = new XmlTextWriter (Console.Out);
writer.Formatting = Formatting.Indented;
writer.Indentation=2;
t.Transform (doc, null, writer );
```

- XSLT может быть выполнено над Dataset'ами, хранящими XmlDataDocument

Литература к лекции

- Б. Трэвис "XML и SOAP. Программирование для серверов BizTalk", М.: "Русская редакция", 2001. 496 с.
- М. Янг "XML. Шаг за шагом", М. ЭКОМ, 2000. 284 с.
- Ф. Бумфрей, О. Диренцо, Й. Даккет и др. "XML. Новые перспективы WWW", М.: ДМК, 2000. 688 с.
- A. Skonnard ".NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation", MSDN Magazine, January 2001. P. 56-71.