

Разработка .NET приложений

●

Лекция 15
LINQ

Сегодня

- Parallel LINQ
- Language Integrated Query (LINQ)
 - LINQ to XML

Parallel LINQ

Parallel LINQ

- Выполнение запроса сразу на всех процессорах и ядрах
- Только LINQ to Objects
- `IEnumerable<T>.AsParallel()` - выполнение операций с последовательностью сразу на всех процессорах и ядрах
 - Не сохраняет порядок последовательности

```
ParallelQuery<Complex> complexes = complexList.AsParallel();  
var result = complexes.Where(...);
```

Добавлены все расширения, аналогичные `IEnumerable<T>`, принимающие и возвращающие `ParallelQuery<T>`
- `ParallelQuery<T>.AsOrdered()` – заставляет сохранять порядок последовательности. Дополнительные расходы на синхронизацию

```
ParallelQuery<Complex> complexes = complexList.AsParallel().AsOrdered().Where(...);
```
- `ParallelQuery<T>.ForAll(Action<T>)` – Выполняет параллельно действие `Action<T>` над каждым элементом последовательности
 - ```
complexList.AsParallel().ForAll(c => c.Re *= 2);
```

# Демонстрация

---

Parallel LINQ

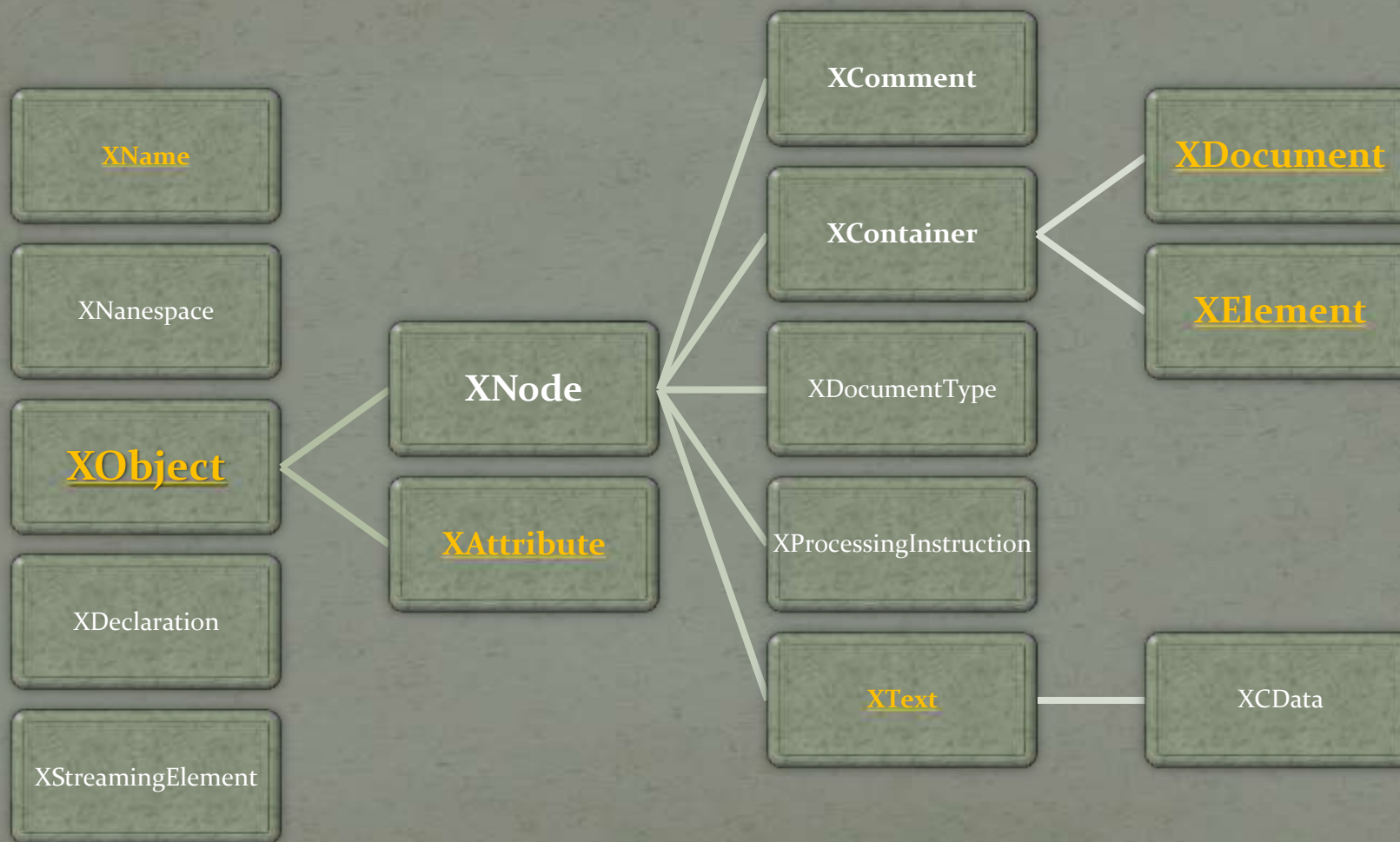
# LINQ to XML

---

# LINQ to XML

- Представляет новый интерфейс работы с XML
- В основу старого интерфейса положен XmlDocument (весь XML документ)
  - Для работы с XML всегда нужно создавать XmlDocument и с ним работать
  - Создать остальные типы можно было только используя XmlDocument
- В основу нового интерфейса положен XElement (элемент, а не документ)
- Упрощен механизм создания и оперирования с различными частями XML
  - Все типы имеют публичные конструкторы

# Объектная модель



Пространство имен System.Xml.Linq



# Создание элемента

- Несколько публичных конструкторов:
  - XElement(XName name);
  - XElement(XName name, params object[] content);
  - XElement(XElement other);

```
XElement students = new XElement("Students",
 new XElement("Student",
 new XAttribute("Year", 2),
 new XElement("FirstName", "Василий"),
 new XElement("LastName", "Петров")),
 new XElement("Student",
 new XComment("Отличник"),
 new XAttribute("Year", 3),
 new XElement("FirstName", "Андрей"),
 new XElement("LastName", "Сидоров")));
```

```
Console.WriteLine(students);
```

- Строка в качестве имени объекта автоматически конвертируется в XName
- Строка в качестве значения элемента автоматически конвертируется в XText

```
<Students>
 <Student Year="2">
 <FirstName>Василий</FirstName>
 <LastName>Петров</LastName>
 </Student>
 <Student Year="3">
 <!--Отличник-->
 <FirstName>Андрей</FirstName>
 <LastName>Сидоров</LastName>
 </Student>
</Students>
```

# Создание элемента

- Трактовка объектов при добавлении их в элемент

string	Автоматически преобразуется в XText
XText	Текстовое содержимое элемента
XCDATA	Добавляется как CData
XElement	Дочерний элемент
XAttribute	Атрибут элемента
XComment	Комментарий
IEnumerable	Каждый элемент последовательности обрабатывается и добавляется отдельно
null	Нет дочернего содержимого
Любой (почти) прочий тип	Вызывается ToString() и трактуется как строка

# Создание атрибута

- Атрибут представляет собой пару “имя-значение”
- Несколько публичных конструкторов:
  - `XAttribute(XAttribute other);`
  - `XAttribute(XName name, object value);`
- Можно создавать сразу добавляя к элементу

```
XElement students = new XElement("Students",
 new XElement("Student",
 new XAttribute("Year", 2),
 new XElement("FirstName", "Василий"),
 new XElement("LastName", "Петров")),
 new XElement("Student",
 new XComment("Отличник"),
 new XAttribute("Year", 3),
 new XElement("FirstName", "Андрей"),
 new XElement("LastName", "Сидоров")));
```

```
Console.WriteLine(students);
```

- Строка в качестве имени атрибута автоматически конвертируется в `XName`

```
<Students>
 <Student Year="2">
 <FirstName>Василий</FirstName>
 <LastName>Петров</LastName>
 </Student>
 <Student Year="3">
 <!--Отличник-->
 <FirstName>Андрей</FirstName>
 <LastName>Сидоров</LastName>
 </Student>
</Students>
```

# Создание комментария

- Представляет текстовое значение – комментарий XML
- Несколько публичных конструкторов:
  - XComment(string value);
  - XComment(XComment other);
- Можно создавать сразу добавляя к элементу

```
XElement students = new XElement("Sudents",
 new XElement("Student",
 new XAttribute("Year", 2),
 new XElement("FirstName", "Василий"),
 new XElement("LastName", "Петров")),
 new XElement("Student",
 new XComment("Отличник"),
 new XAttribute("Year", 3),
 new XElement("FirstName", "Андрей"),
 new XElement("LastName", "Сидоров")));
```

```
Console.WriteLine(students);
```

```
<Sudents>
 <Student Year="2">
 <FirstName>Василий</FirstName>
 <LastName>Петров</LastName>
 </Student>
 <Student Year="3">
 <!--Отличник-->
 <FirstName>Андрей</FirstName>
 <LastName>Сидоров</LastName>
 </Student>
</Sudents>
```

# Создание документа

- Несколько публичных конструкторов:
  - XmlDocument();
  - XmlDocument(params object[] content);
  - XmlDocument(XDeclaration declaration, params object[] content);

```
XmlDocument doc = new XmlDocument(new XDeclaration("1.0", "UTF-8", "yes"),
 new XElement("Students", null));
Console.WriteLine(doc);
```

```
<Students />
```

- Объявление XML
  - Класс XDeclaration
  - Может добавляться только к документу
  - XDeclaration(string version, string encoding, string standalone);

# XName и XNamespace

- XName
  - Не имеет публичных конструкторов. Автоматически конвертируется из строки
    - XName name = "Students";
  - Состоит из LocalName – имени и пространства имен Namespace
- XNamespace – представляет пространство имен
  - Не имеет публичных конструкторов. Автоматически конвертируется из строки
    - XNamespace aw = "http://www.adventure-works.com";

```
XNamespace aw = "http://www.adventure-works.com";
XElement root = new XElement(aw + "Root",
 new XAttribute("xmlns", "http://www.adventure-works.com"),
 new XElement(aw + "Child", "content"));
Console.WriteLine(root);
```

```
<Root xmlns="http://www.adventure-works.com">
 <Child>content</Child>
</Root>
```

# Демонстрация

---

Создание XML

# Ввод / вывод XML

- Статический метод **Load()**
  - Загружает XML из файла / потока
  - Реализован у XmlDocument и XElement  
`XmlDocument document = XmlDocument.Load("students.xml");`
- Статический метод **Parse()**
  - Загружает XML из строки
  - Реализован у XmlDocument и XElement  
`XElement students = XElement.Parse("<Students><Student Year=\\\"3\\\"/></Students>");`
- **Save()**
  - Сохраняет XML в файл / поток
  - Реализован у XmlDocument и XElement  
`students.Save("students.xml");`
- **ToString()**
  - Переопределен у всех XElement
  - Возвращает форматированное содержание XML  
`Console.WriteLine(students);`



# Обход деревьев XML

- XContainer.**Element()** – возвращает первый дочерний элемент с указанным именем
- XContainer.**Elements()** – возвращает все дочерние элементы
- XContainer.**Nodes()** – возвращает все дочерние узлы
  
- XmlNode.**NextNode** / XmlNode.**PreviousNode** – свойства содержат предыдущий / следующий узел в дереве
  
- XObject.**Document** – ссылка на документ
- XObject.**Parent** – ссылка родителя в XML
  
- XmlNode.**Ancestors()** – возвращает все родительские элементы (рекурсивный обход XML вверх)
- XElement.**AncestorsAndSelf()** – возвращает все родительские элементы (рекурсивный обход XML вверх) и себя
  
- XContainer.**Descendants()** – рекурсивно возвращает все дочерние элементы
- XElement.**DescendantsAndSelf()** – рекурсивно возвращает все дочерние элементы и сам элемент
  
- XmlNode.**NodesAfterSelf()** / XmlNode.**NodesBeforeSelf()** – возвращает все узлы после / перед текущим
- XmlNode.**ElementsAfterSelf()** / XmlNode.**ElementsBeforeSelf()** – возвращает все элементы после / перед текущим

# Изменения XML

- Добавление узлов
  - `XContainer.Add()` – добавляет узел в конец документа или элемента
  - `XContainer.AddFirst()` – добавляет узел в начало документа или элемента
  - `XNode.AddBeforeSelf()` – добавляет узел перед текущим узлом
  - `XNode.AddAfterSelf()` – добавляет узел после текущего узла
- Удаление узлов
  - `XNode.Remove()` – удаление текущего узла
  - `IEnumerable<T>.Remove()` – удаление нескольких узлов
  - `XElement.RemoveAll()` – удаление всего содержимого элемента, но не сам элемент
- Обновление узлов
  - `XElement.Value` – задает текстовое значение элемента
  - `XElement.SetElementValue()` – обновляет указанный дочерний элемент
    - создает если такого нет
    - изменяет, если такой есть
    - удаляет, если значение установлено в null
  - `XElement.ReplaceAll()` – заменяет все поддерево элемента на новое

# Атрибуты

- Обход атрибутов:
  - XElement.FirstAttribute, XElement.LastAttribute – первый / последний атрибуты элемента
  - XElement.NextAttribute, XElement.PreviousAttribute – следующий / предыдущий атрибуты элемента
  - XElement.**Attribute()** – возвращает атрибут с указанным именем
  - XElement.**Attributes()** – возвращает все атрибуты элемента
- Добавления атрибутов
  - Также как и добавление узлов
- Удаление атрибутов
  - XAttribute.Remove() – удаление текущего атрибута
  - IEnumerable<T>.Remove() – удаление коллекции атрибутов
- Изменения атрибутов
  - XAttribute.Value – значение атрибута

# Получение значений

- Для получение значений элементов и атрибутов достаточно просто привести элемент или атрибут к нужному типу

```
XElement firstName = new XElement("FirstName", "Василий");
Console.WriteLine(firstName);
Console.WriteLine((string)firstName);
```

```
<FirstName>Василий</FirstName>
Василий
```

```
XAttribute year = new XAttribute("Year", 3);
Console.WriteLine(year);
Console.WriteLine((int)year);
```

```
Year="3"
3
```

# Расширения

- Многие методы работы с XML могут быть вызваны и на последовательности:
- **Elements**
  - `IEnumerable<XElement> Elements<T>(this IEnumerable<T> source) where T : XElement;`
  - `IEnumerable<XElement> Elements<T>(this IEnumerable<T> source, XElement name) where T : XElement;`

```
IEnumerable<XElement> names = students.Elements("Student").Elements("FirstName");
foreach (XElement firstName in names) Console.WriteLine((string)firstName);
```

- **Nodes**
  - `IEnumerable<XNode> Nodes<T>(this IEnumerable<T> source) where T : XElement;`
- **Attributes**
  - `IEnumerable<XAttribute> Attributes(this IEnumerable<XElement> source);`
  - `IEnumerable<XAttribute> Attributes(this IEnumerable<XElement> source, XElement name);`

```
IEnumerable<XAttribute> years = students.Elements("Student").Attributes("Year");
foreach (XAttribute year in years) Console.WriteLine((int)year);
```

# Расширения

- **Ancestors**

- `IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source) where T : XElement;`
- `IEnumerable<XElement> Ancestors<T>(this IEnumerable<T> source, XName name) where T : XElement;`
- `IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source);`
- `IEnumerable<XElement> AncestorsAndSelf(this IEnumerable<XElement> source, XName name);`

- **Descendants**

- `IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source) where T : XElement;`
- `IEnumerable<XElement> Descendants<T>(this IEnumerable<T> source, XName name) where T : XElement;`

```
XElement sidorov = students.Descendants("Student")
 .Where(e => (string)e.Element("LastName") == "Сидоров")
 .FirstOrDefault();
Console.WriteLine(sidorov);
```

```
IEnumerable<XElement> petrov = from s in students.Descendants("Student")
 where (string)s.Element("LastName") == "Сидоров"
 select s;
Console.WriteLine(petrov.Single());
```

- `IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source);`
- `IEnumerable<XElement> DescendantsAndSelf(this IEnumerable<XElement> source, XName name);`
- `IEnumerable<XNode> DescendantNodes<T>(this IEnumerable<T> source) where T : XElement;`
- `IEnumerable<XNode> DescendantNodesAndSelf(this IEnumerable<XElement> source);`

- **Удаление узлов и атрибутов**

- `void Remove<T>(this IEnumerable<T> source) where T : XElement;`
- `void Remove(this IEnumerable<XAttribute> source);`

- **Сортировка в порядке присутствия в документе**

- `IEnumerable<T> InDocumentOrder<T>(this IEnumerable<T> source) where T : XElement;`

# Демонстрация

---

Запросы к XML

# Дополнительные возможности

- Проверка достоверности XML документа
- Получение и работа с XSD схемой
- XSLT преобразование
- Использование XPath для навигации по XML