LINQ

Основы работы с Language-Integrated Query

LINQ. Определение.

- **LINQ** (Language-Integrated Query) представляет простой и удобный ***язык запросов к источнику данных***. В качестве источника данных может выступать объект, реализующий интерфейс **IEnumerable** (например, стандартные коллекции, массивы), набор данных **DataSet**, документ **XML**. Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.
- Пространство имен – **System.Linq;**

LINQ. Разновидности.

- **LINQ to Objects**: применяется для работы с массивами и коллекциями
- **LINQ to Entities**: используется при обращении к базам данных через технологию Entity Framework
- **LINQ to Sql**: технология доступа к данным в MS SQL Server
- **LINQ to XML**: применяется при работе с файлами XML
- **LINQ to DataSet**: применяется при работе с объектом DataSet
- **Parallel LINQ (PLINQ)**: используется для выполнения параллельных запросов

Пример без LINQ

```
static void Main()
{
    string[] names = { "Андрей", "Иван", "Алексей", "Дмитрий", "Никита", "Александр" };

    var selectedNames = new List<string>();
    foreach (string s in names)
    {
        if (s.ToUpper().StartsWith("A"))
        {
            selectedNames.Add(s);
        }
    }

    selectedNames.Sort();

    foreach (string s in selectedNames)
    {
        Console.WriteLine(s);
    }
    Console.ReadLine();
}
```

Пример с LINQ

```
static void Main()
{
    string[] names = { "Андрей", "Иван", "Алексей", "Дмитрий", "Никита", "Александр" };

    var selectedNames = from t in names // определяем каждый объект из names как t
                        where t.ToUpper().StartsWith("A") //фильтрация по критерию
                        orderby t  // упорядочиваем по возрастанию
                        select t; // выбираем объект

    foreach (string s in selectedNames)
    {
        Console.WriteLine(s);
    }
    Console.ReadLine();
}
```

Пример с LINQ (запись в одну строку)

```
static void Main()
{
    string[] names = { "Андрей", "Иван", "Алексей", "Дмитрий", "Никита", "Александр" };

    var selectedNames = from t in names where t.ToUpper().StartsWith("A") orderby t select t;

    foreach (string s in selectedNames)
    {
        Console.WriteLine(s);
    }
    Console.ReadLine();
}
```

LINQ-запрос. Простейшее определение.

from *переменная* **in** *набор_объектов* **select** *переменная*;

```
var selectedNames = from t in names where t.ToUpper().StartsWith("A") orderby t select t;
```

- Выражение **from t in names** проходит по всем элементам массива names и определяет каждый элемент как **t**. Используя переменную **t** мы можем проводить над ней разные операции.
- Далее с помощью оператора **where** проводится фильтрация объектов, и если объект соответствует критерию (в данном случае начальная буква должна быть "A"), то этот объект передается дальше.
- Оператор **orderby** упорядочивает по возрастанию, то есть сортирует выбранные объекты.
- Оператор **select** передает выбранные значения в результирующую выборку, которая возвращается LINQ-выражением.

LINQ. Методы расширения.

Кроме стандартного синтаксиса **from .. in .. select** для создания запроса LINQ мы можем применять специальные методы расширения, которые определены в интерфейсе **IEnumerable**. Как правило, эти методы реализуют ту же функциональность, что и операторы LINQ типа **where** или **orderby**.

LINQ. Методы расширения. Пример.

```
var selectedNames = from t in names where t.ToUpper().StartsWith("A") orderby t select t;
```

```
static void Main()
{
    string[] names = { "Андрей", "Иван", "Алексей", "Дмитрий", "Никита", "Александр" };

    var selectedNames = names.Where(t => t.ToUpper().StartsWith("A")).OrderBy(t => t);

    foreach (string s in selectedNames)
    {
        Console.WriteLine(s);
    }
    Console.ReadLine();
}
```

LINQ. Сочетание подходов.

- используем стандартный синтаксис **LINQ** и метод расширения **Count**(), возвращающий количество элементов в выборке:

```
int number = (from t in names where t.ToUpper().StartsWith("A") select t).Count();
```

LINQ. Используемые методы расширения

- **Select**: определяет проекцию выбранных значений
- **Where**: определяет фильтр выборки
- **OrderBy**: упорядочивает элементы по возрастанию
- **OrderByDescending**: упорядочивает элементы по убыванию
- **Join**: соединяет две коллекции по определенному признаку
- **GroupBy**: группирует элементы по ключу
- **Reverse**: располагает элементы в обратном порядке
- **Contains**: определяет, содержит ли коллекция определенный элемент
- **Distinct**: удаляет дублирующиеся элементы из коллекции
- **Count**: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию
- **Concat**: объединяет две коллекции
- **First**: выбирает первый элемент коллекции
- **ElementAt**: выбирает элемент последовательности по определенному

LINQ. Методы расширения. Пример со словарём.

```
static void Main()
{
    Dictionary<int, string> numbers = new Dictionary<int, string>()
    {
        {1, "Один"},
        {2, "Два"},
        {3, "Три"},
        {4, "Четыре"}
    };

    var selectedNumber = numbers.First(f => f.Key > 2).Value; //вернет "Три"
}
```

LINQ. Методы расширения. Пример с XML.

```
<?xml version="1.0"?>
- <PurchaseOrder OrderDate="1999-10-20" PurchaseOrderNumber="99503">
  - <Address Type="Shipping">
      <Name>Ellen Adams</Name>
      <Street>123 Maple Street</Street>
      <City>Mill Valley</City>
      <State>CA</State>
      <Zip>10999</Zip>
      <Country>USA</Country>
    </Address>
  - <Address Type="Billing">
      <Name>Tai Yee</Name>
      <Street>8 Oak Avenue</Street>
      <City>Old Town</City>
      <State>PA</State>
      <Zip>95819</Zip>
      <Country>USA</Country>
    </Address>
    <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
</PurchaseOrder>
```

LINQ. Методы расширения. Пример с XML.

```
static void Main()
{
    XElement xElement = XElement.Load("11.xml");

    var address = xElement.Elements().First(e => e.Name == "Address" && e.Attribute("Type").Value == "Billing");
}
```