16. LINQ

§16.1 Основы LINQ

LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных. В качестве источника данных могут выступать стандартные коллекции, массивы, документ XML. Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.

Существует несколько разновидностей LINQ:

1. LINQ to Objects: применяется для работы с массивами и коллекциями
2. LINQ to Entities: используется при обращении к базам данных через технологию Entity Framework
3. LINQ to Sql: технология доступа к данным в MS SQL Server
4. LINQ to XML: применяется при работе с файлами XML
5. LINQ to DataSet: применяется при работе с объектом DataSet

Для использования функциональности LINQ необходимо подключать пространство имен `System.LINQ`.

В чем же удобство LINQ? Посмотрим на простейшем примере. Выберем из массива строки, начинающиеся на определенную букву и отсортируем полученный список:

Листинг 16.1

```
1  static void Main()
2  {
3  string[] names = { "Айгуль", "Алиса", "Борис", "Константин", "Абдулла", "Татьяна" };
4  var selectedNames = new List<string>();
5  foreach (string s in names)
6  {
7  if (s.ToUpper().StartsWith("A"))
8  selectedNames.Add(s);
9  }
10 selectedNames.Sort();
11 foreach (string s in selectedNames)
12 Console.WriteLine(s);
13 }
```

Теперь проведем те же действия с помощью LINQ:

Листинг 16.2

```
1  string[] names = { "Айгуль", "Алиса", "Борис", "Константин", "Абдулла", "Татьяна" };
2  var selectedNames = from n in names // определяем каждый объект из names как n
3  where n.ToUpper().StartsWith("A") //фильтрация по критерию
4  orderby n  // упорядочиваем по возрастанию
5  select n; // выбираем объект
6  foreach (string s in selectedNames)
7  Console.WriteLine(s);
```

Простейшее определение запроса LINQ выглядит следующим образом:

```
from переменная in набор_объектов
select переменная;
```

Выражение `from n in names` проходит по всем элементам массива `names` и определяет каждый элемент как `n`. Используя переменную `n` мы можем проводить над ней разные операции.

Несмотря на то, что мы не указываем тип переменной `n`, выражения LINQ являются строго типизированными. То есть среда автоматически распознает, что набор `names` состоит из объектов `string`, поэтому переменная `n` будет рассматриваться в качестве строки.

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.

В данном случае результатом выражения LINQ является объект `IEnumerable<T>`. Нередко результирующая выборка определяется с помощью ключевого слова `var`, тогда компилятор на этапе компиляции сам выводит тип.

Преимуществом подобных запросов также является и то, что они интуитивно похожи на запросы языка SQL, хотя и имеют некоторые отличия.

§16.2 Методы расширения LINQ

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

Например:

Листинг 16.3

```
1  string[] names = { "Айгуль", "Алиса", "Борис", "Константин", "Абдулла", "Татьяна" };
2  var selectedNames = names.Where(n=>n.ToUpper().StartsWith("A")).OrderBy(n => n);
3  foreach (string s in selectedNames)
4      Console.WriteLine(s);
```

Запрос

```
teams.Where(n=>n.ToUpper().StartsWith("Б")).OrderBy(n => n)
```

будет аналогичен предыдущему. Он состоит из цепочки методов `Where` и `OrderBy`. В качестве аргумента эти методы принимают делегат или лямбда-выражение.

Не каждый метод расширения имеет аналог среди операторов LINQ, но в этом случае можно сочетать оба подхода. Например, используем стандартный синтаксис linq и метод расширения `Count()`, возвращающий количество элементов в выборке:

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

Список используемых методов расширения LINQ:

Таблица 16.1 – Методы расширения LINQ

Метод	Описание
`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

Метод	Описание
All	определяет, все ли элементы коллекции удовлетворяют определенному условию
Any	определяет, удовлетворяет хотя бы один элемент коллекции определенному условию
Contains	определяет, содержит ли коллекция определенный элемент
Distinct	удаляет дублирующиеся элементы из коллекции
Except	возвращает разность двух коллекцию, то есть те элементы, которые создаются только в одной коллекции
Union	объединяет две однородные коллекции
Intersect	возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях
Count	подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию
Sum	подсчитывает сумму числовых значений в коллекции
Average	подсчитывает среднее значение числовых значений в коллекции
Min	находит минимальное значение
Max	находит максимальное значение
Take	выбирает определенное количество элементов
Skip	пропускает определенное количество элементов
TakeWhile	возвращает цепочку элементов последовательности, до тех пор, пока условие истинно
SkipWhile	пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы
Concat	объединяет две коллекции
Zip	объединяет две коллекции в соответствии с определенным условием

Метод	Описание
`First`	выбирает первый элемент коллекции
`Single`	выбирает единственный элемент коллекции, если коллекция содержит больше или меньше одного элемента, то генерируется исключение
`ElementAt`	выбирает элемент последовательности по определенному индексу
`Last`	выбирает последний элемент коллекции

§16.3 Фильтрация выборки и проекция

Фильтрация

Для выбора элементов из некоторого набора по условию используется метод `Where`. Например, выберем все четные элементы, которые больше 10.

Фильтрация с помощью операторов LINQ:

Листинг 16.4

```
1  int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 };
2  IEnumerable<int> evens = from i in numbers
3                           where i%2==0 && i>10
4                           select i;
5  foreach (int i in evens)
6      Console.WriteLine(i);
```

Здесь используется конструкция `from`: `from i in numbers`.

Тот же запрос с помощью метода расширения:

Листинг 16.5

```
1  int[] numbers = { 1, 2, 3, 4, 10, 34, 55, 66, 77, 88 }
2  IEnumerable<int> evens = numbers.Where(i => i % 2 == 0 && i > 10)
```

Если выражение в методе `Where` для определенного элемента будет равно `true` (в данном случае выражение `i % 2 == 0 && i > 10`), то данный элемент попадает в результирующую выборку.

Выборка сложных объектов

Допустим, у нас есть класс пользователя:

Листинг 16.6

```
1  class User
```

```
2   {
3       public string Name { get;set; }
4       public int Age { get; set; }
5       public List<string> Languages { get; set; }
6       public User()
7       {
8           Languages = new List<string>();
9       }
10  }
```

Создадим набор пользователей и выберем из них тех, которым больше 25 лет:

Листинг 16.7

```
1   List<User> users = new List<User>
2   {
3   new User {Name="Том", Age=23, Languages = new List<string>
    {"английский", "немецкий" }},
4   new User {Name="Боб", Age=27, Languages = new List<string>
    {"английский", "французский" }},
5   new User {Name="Джон", Age=29, Languages = new List<string>
    {"английский", "испанский" }},
6   new User {Name="Элис", Age=24, Languages = new List<string>
    {"испанский", "немецкий" }}
7   };
8   var selectedUsers = from user in users
9                       where user.Age > 25
10                      select user;
11  foreach (User user in selectedUsers)
12      Console.WriteLine($"{user.Name} - {user.Age}");
```

Аналогичный запрос с помощью метода расширения `Where`:

```
var selectedUsers = users.Where(u=> u.Age > 25);
```

Сложные фильтры

Теперь рассмотрим более сложные фильтры. Например, в классе пользователя есть список языков, которыми владеет пользователь. Что если нам надо отфильтровать пользователей по языку:

Листинг 16.8

```
1   var selectedUsers = from user in users
2   from lang in user.Languages
3   where user.Age < 28
4   where lang == "английский"
5   select user;
```

Для создания аналогичного запроса с помощью методов расширения применяется метод SelectMany:

Листинг 16.9

```
1  var selectedUsers = users.SelectMany(u => u.Languages,
2  (u, l) => new { User = u, Lang = l })
3  .Where(u => u.Lang == "английский" && u.User.Age < 28)
4  .Select(u=>u.User);
```

Метод SelectMany() в качестве первого параметра принимает последовательность, которую надо проецировать, а в качестве второго параметра - функцию преобразования, которая применяется к каждому элементу. На выходе она возвращает 8 пар "пользователь - язык"

```
(new { User = u, Lang = l }),
```

к которым потом применяется фильтр с помощью Where.

Проекция

Проекция позволяет спроектировать из текущего типа выборки какой-то другой тип. Для проекции используется оператор select. Допустим, у нас есть набор объектов следующего класса, представляющего пользователя:

Листинг 16.10

```
1  class User
2  {
3      public string Name { get;set; }
4      public int Age { get; set; }
5  }
```

Но нам нужен не весь объект, а только его свойство Name:

Листинг 16.11

```
1  List<User> users = new List<User>();
2  users.Add(new User { Name = "Sam", Age = 43 });
3  users.Add(new User { Name = "Tom", Age = 33 });
4  var names = from u in users select u.Name;
5  foreach (string n in names)
6       Console.WriteLine(n);
```

Результат выражения LINQ будет представлять набор строк, поскольку выражение select u.Name выбирают в результирующую выборку только значения свойства Name.

Аналогично можно создать объекты другого типа, в том числе анонимного:

Листинг 16.12

```
List<User> users = new List<User>();
users.Add(new User { Name = "Sam", Age = 43 });
users.Add(new User { Name = "Tom", Age = 33 });
var items = from u in users
            select new
            {
                FirstName = u.Name,
                DateOfBirth = DateTime.Now.Year - u.Age
            };
foreach (var n in items)
Console.WriteLine($"{n.FirstName} - {n.DateOfBirth}");
```

Здесь оператор `select` создает объект анонимного типа, используя текущий объект `User`. И теперь результат будет содержать набор объектов данного анонимного типа, в котором определены два свойства: `FirstName` и `DateOfBirth`.

В качестве альтернативы мы могли бы использовать метод расширения `Select()`:

Листинг 16.13

```
// выборка имен
var names = users.Select(u => u.Name);
// выборка объектов анонимного типа
var items = users.Select(u => new
{
    FirstName = u.Name,
    DateOfBirth = DateTime.Now.Year - u.Age
});
```

Выборка из нескольких источников

В LINQ можно выбирать объекты не только из одного, но и из большего количества источников.

Например, возьмем классы:

Листинг 16.14

```
class Phone
{
    public string Name { get; set; }
    public string Company { get; set; }
}
class User
```

```
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Создадим два разных источника данных и произведем выборку:

Листинг 16.15

```
List<User> users = new List<User>()
{
    new User { Name = "Sam", Age = 43 },
    new User { Name = "Tom", Age = 33 }
};
List<Phone> phones = new List<Phone>()
{
    new Phone {Name="Lumia 630", Company="Microsoft" },
    new Phone {Name="iPhone 6", Company="Apple"},
};
var people = from user in users
             from phone in phones
             select new { Name = user.Name, Phone = phone.Name };
foreach (var p in people)
    Console.WriteLine($"{p.Name} - {p.Phone}");
```

Таким образом, при выборке из двух источников каждый элемент из первого источника будет сопоставляться с каждым элементом из второго источника. То есть должно получиться четыре пары.