Лабораторная работа №7

Язык запросов LINQ

1) Введение

LINQ – это набор появившихся в Visual Studio 2008 функций, который значительно расширяет возможности синтаксиса языка C#. LINQ предоставляет стандартные, простые в изучении шаблоны для запроса и изменения данных и технологии, которые могут быть расширены для поддержки практически любого типа источника данных.

В C# можно писать запросы LINQ для обращения к базам данных SQL Server, XML-документам, наборам данных ADO.NET и к любым коллекциям объектов, поддерживающим интерфейс IEnumerable или универсальный интерфейс IEnumerable<T>. Также планируется поддержка LINQ для ADO.NET Entity Framework, а сторонние разработчики пишут поставщики LINQ для многих веб-служб и других реализаций баз данных.

2) Запросы в LINQ

Запрос представляет собой выражение, извлекающее данные из источника данных. Запросы обычно выражаются на специальном языке запросов. Со временем были разработаны различные языки для различных типов источников данных, например SQL для реляционных баз данных и XQuery для XML. Таким образом, разработчики вынуждены изучать новый язык запросов для каждого типа источника данных или формата данных, который они должны поддерживать. LINQ упрощает ситуацию, предлагая единообразную модель для работы с данными в различных видах источников и форматов данных. В запросе LINQ работа всегда осуществляется с объектами. Для запросов и преобразований данных в XML-документах, базах данных SQL, наборах данных ADO.NET, коллекциях .NET и любых других форматах, для

которых доступен поставщик LINQ, используются одинаковые базовые шаблоны кодирования. Все операции запроса LINQ состоят из трех различных действий.

1) Получение источника данных.

2) Создание запроса.

3) Выполнение запроса.

В следующем примере показано выражение этих трех частей операции запроса в исходном коде. В примере в качестве источника данных для удобства используется массив целых чисел; тем не менее, те же принципы применимы и к другим источникам данных.

```
class IntroToLINQ
{
    static void Main()
    {
        // Три части запроса в LINQ
        // 1. Источник данных.
        int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

        // 2. Создание запроса.
        // numQuery реализует интерфейс IEnumerable<int>
        var numQuery =
            from num in numbers
            where (num % 2) == 0
            select num;

        // 3. Выполнение запроса.
        foreach (int num in numQuery)
        {
            Console.Write("{0,1} ", num);
        }
    }
}
```

На рисунке 1 показана завершенная операция запроса. В LINQ выполнение запроса отличается от самого запроса; другими словами, создание переменной запроса само по себе не связано с получением данных.

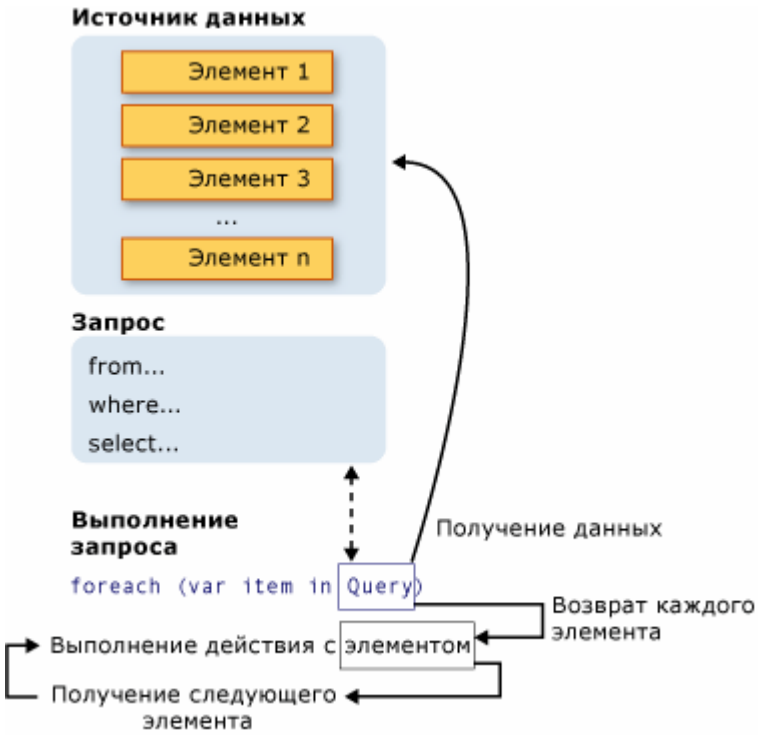

Рисунок 1 – Иллюстрация к запросу в LINQ

В предыдущем примере источником данных является массив, поэтому он неявно поддерживает универсальный интерфейс IEnumerable<T>. Это значит, что к нему можно выполнять запросы с LINQ. Запрос выполняется в инструкцию foreach, и foreach для IEnumerable или IEnumerable<T>. Типы, которые поддерживают IEnumerable<T> или производный интерфейс как универсальный шаблон IQueryable<T> вызываются запрашиваемыми типами. Запрос указывает, какую информацию нужно извлечь из источника или источников данных. При необходимости, запрос также указывает способ сортировки, группировки и формирования этих сведений перед возвращением. Запрос хранится в переменной запроса и инициализируется выражением запроса. Чтобы упростить написание запросов, в C# появился новый синтаксис запроса.

Как уже говорилось ранее, сама переменная запроса только хранит команды запроса. Фактическое выполнение запроса откладывается до выполнения итерации переменной запроса в операторе foreach. Эту концепцию называют отложенным выполнением.

3) Основные операции запросов LINQ

Получение источника данных:

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

Переменная диапазона (cust) схожа с переменной итерации в цикле foreach за исключением того, что в выражении запроса не происходит фактической итерации. При выполнении запроса переменная диапазона будет использоваться как ссылка на каждый последующий элемент в customers. Поскольку компилятор может определить тип cust, нет необходимости указывать его в явном виде (поэтому используется ключевое слово var).

Фильтрация:

Возможно, наиболее распространенной операцией запроса является применение фильтра в виде логического выражения. Фильтр приводит к возвращению запросом только тех элементов, для которых выражение является истинным. Результат создается с помощью предложения where. Фильтр фактически указывает элементы для исключения из исходной последовательности. В следующем примере возвращаются только customers, находящиеся в Лондоне.

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

Для применения нужного числа выражений фильтра в предложении where можно использовать знакомые логические операторы C# AND и OR. Например, для получения только заказчиков из Лондона AND с именем Devon следует написать следующий код.

```
where cust.City == "London" && cust.Name == "Devon"
```

Сортировка:

Часто целесообразно отсортировать возвращенные данные. Предложение orderby сортирует элементы возвращаемой последовательности в зависимости от компаратора по умолчанию для сортируемого типа. Например, следующий запрос может быть расширен для сортировки результатов на основе свойства

Name. Поскольку Name является строкой, сравнение по умолчанию выполняется в алфавитном порядке от А до Я.

```
var queryLondonCustomers3 = from cust in customers
                            where cust.City == "London"
                            orderby cust.Name ascending
                            select cust;
```

Группировка:

Предложение group позволяет группировать результаты на основе указанного ключа. Например, можно указать, что результаты должны быть сгруппированы по City так, чтобы все заказчики из Лондона или Парижа оказались в отдельных группах. В этом случае ключом является cust.City.

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
var queryCustomersByCity =
       from cust in customers
       group cust by cust.City;

// customerGroup is an IGrouping<string, Customer>
foreach (var customerGroup in queryCustomersByCity)
{
    Console.WriteLine(customerGroup.Key);
    foreach (Customer customer in customerGroup)
    {
        Console.WriteLine("    {0}", customer.Name);
    }
}
```

Когда запрос завершается предложением group, результаты представляются в виде списка из списков. Каждый элемент в списке является объектом, имеющим член Key и список элементов, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл foreach. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл – итерацию членов каждой группы.

Если необходимо ссылаться на результаты операции группировки, можно использовать ключевое слово into для создания идентификатора, который можно будет запрашивать. Следующий запрос возвращает только те группы, которые содержат более двух заказчиков.

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
       from cust in customers
       group cust by cust.City into custGroup
       where custGroup.Count() > 2
       orderby custGroup.Key select custGroup;
```

Выбор (Проецирование):

Предложение select создает результаты запроса и задает форму или тип каждого возвращаемого элемента. Например, можно указать, будут ли результаты состоять из полных объектов Customer, только из одного члена, подмножества членов или некоторых совершенно других типов, на основе вычислений или создания новых объектов. Когда предложение select создает что-либо отличное от копии исходного элемента, операция называется проекцией. Использование проекций для преобразования данных является мощной возможностью выражений запросов LINQ.

4) Задания к лабораторной работе №7

Для своего варианта заданий по лабораторной работе №2 создать список объектов реализованных классов. Реализовать несколько запросов LINQ к этому списку, используя операции фильтрации, сортировки и группировки. Например, для варианта 1 (Класс время) можно создать запросы получения всех временных промежутков в порядке возрастания, группировки интервалов по часам, получения временных промежутков до полудня и после полудня и т. д.

5) Контрольные вопросы

1) Что такое LINQ и для чего он предназначен?
2) Когда происходит выполнение запроса?
3) Для чего используется предложение where?
4) Как отсортировать результаты запроса?
5) Что позволяет сделать предложение group?