Использование технологии LINQ

11

Технология LINQ (Language Integrated Query) предназначена для обработки (для организации запросов и преобразований) практически любого источника данных, начиная от массивов, файлов, строк, коллекций объектов .NET Framework, баз данных SQL Server, наборов данных ADO.NET (DataSet) и XML-документов. LINQ упрощает ситуацию, предлагая стандартные шаблоны для работы с данными в различных видах источников и различных форматов. Стандартные шаблоны включают в себя основные операции запросов LINQ: фильтрация, упорядочение, группировка, соединение, выбор (проецирование), статистическая обработка. По форме синтаксис языка LINQ очень похож на язык запросов SQL.

Следует отметить, что ту технологию LINQ-запросов со стандартными операторами запросов (from, where, select и др.), которую имеют языки Visual Basic и C#, язык MS Visual C++ для среды .NET *не поддерживает*. Технология LINQ-запросов успешно работает в Visual Basic и C#, начиная с версии Visual Studio 2008. Вероятно, у Microsoft до внедрения такой же технологии и в C++/CLI 2010 еще «не дошли руки». Во время компиляции выражения LINQ-запроса преобразуются в вызовы методов. Это как раз то, что понятно среде CLR .NET — вызовы методов. Поэтому мы можем на языке MS Visual C++ для среды .NET обращаться непосредственно к этим методам технологии LINQ. Таким образом, в данной главе мы продемонстрируем использование синтаксиса LINQ-методов вместо синтаксиса LINQ-запросов.

Пример 89. Манипулирование массивом данных методами класса Linq::Enumerable

Продемонстрируем возможность выборки из массива необходимых элементов и совершения с ними некоторых преобразований с помощью методов класса Linq::Enumerable. Конкретная задача состоит в следующем: мы имеем строковый массив имен людей, из него извлекаем имена длиной шесть символов, записывая их в список (коллекцию). При этом избавляемся от дублирования элементов в списке, сортируем их в алфавитном порядке и переводим все символы в верхний регистр.

Для решения этой задачи запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR, укажем имя Name — LinqМассив. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.1 приведен программный код обсуждаемого консольного приложения.

Листинг 11.1. Извлечение данных из строкового массива

```
// LinqМассив.cpp: главный файл проекта.
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим для краткости выражений:
using namespace System::Linq;
bool Предикат(String ^ S)
{
    // Если число букв равно шести, то возвращаем true:
    bool A = false;
    if (S->Length == 6) A = true;
    return A;
}
String ^ Предикат2(String ^ S)
{
    // Переводим строку в верхний регистр:
    S = S->ToUpper();
    return S;
}
int main(array<System::String ^> ^args)
{
    // Программа в строковом массиве имен выбирает имена, состоящие из
    // шести букв. В списке выбранных имен сортируем их в алфавитном порядке,
    // все строки переводим в верхний регистр и избавляемся от дублирования
    // имен. При этом вместо синтаксиса LINQ-запросов (как в VB и C#)
    // используем синтаксис LINQ-методов
    Console::Title = "Фильтрация массива методами класса Linq::Enumerable";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    auto СтрокаИмен =
        "Витя Лариса Лариса Лена Андрей Женя \n" +
        "Александр Лариса Виктор Света Оксана Наташа";
    // Из строки имен получаем массив имен, задавая в качестве
    // разделителя подстрок символ пробела:
    auto Имена = СтрокаИмен->Split(' ');
    Console::WriteLine("ЗАДАЧА 1. В списке имен:\n\n" + СтрокаИмен);
    Console::WriteLine(
        "\nвыбираем имена с количеством букв равным\n" +
```

продолжение ⇩

Листинг 11.1 (продолжение)

```
        "шести, сортируем полученный список,       \n" +
        "переводим в верхний регистр и избавляемся \n" +
        "от дублирования имен:                     \n");
    // Из массива имен получаем список имен, длина которых - шесть букв:
    auto Запрос = Enumerable::Where<String^>(Имена, gcnew
                                  Func<String ^,bool>(Предикат));
    // Сортируем полученный список в алфавитном порядке:
    Запрос = Enumerable::OrderBy(Запрос, gcnew
                              Func<String ^,String ^>(Предикат2));
    // Переводим в верхний регистр:
    Запрос = Enumerable::Select(Запрос, gcnew
                              Func<String ^,String ^>(Предикат2));
    // Избавляемся от дублирования имен:
    Запрос = Enumerable::Distinct(Запрос);
    // Выводим на консоль результаты:
    for each (String ^ x in Запрос)
                Console::WriteLine("{0} ", x);
    Console::ReadKey();
    return 0;
}
```

Как видно из программного кода, после присвоения массиву Имена начальных значений, используя LINQ-функцию Where, создаем запрос, который предусматривает выбор из массива Имена строк длиной (Length) ровно шесть символов. Последнее условие задается в отдельной процедуре Предикат. Запись выбранных имен осуществляется в список Запрос. Для сортировки списка в алфавитном порядке имен используем LINQ-функцию OrderBy, а для перевода в верхний регистр — Select. Далее для удаления повторяющихся имен в списке используем функцию Distinct. Цикл for each выводит на консоль результат манипуляций с массивом и списком.

На рис. 11.1 приведен фрагмент работы программы.

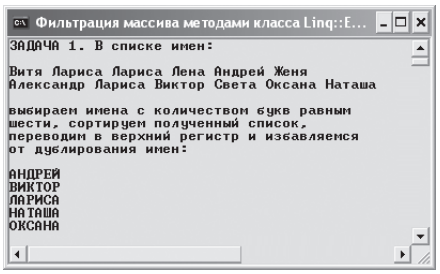

Рис. 11.1. Манипулирование массивом данных с помощью технологии LINQ

Убедиться в работоспособности программы можно, открыв решение LinqМассив.sln в папке LinqМассив.

Пример 90. Запрос к коллекции (списку) данных методами LINQ

В некоторых случаях хранение данных в коллекции (скажем, в списке типа List) может оказаться более эффективным, чем в массиве. Например, если число элементов в массиве изменяется часто или нельзя заранее определить максимальное количество необходимых элементов, то при использовании коллекции можно добиться большей производительности. Но если размер массива не изменяется или изменяется довольно редко, то использование массива приведет к большей эффективности. Как всегда, производительность в большей степени зависит от конкретного приложения. Как советуют в документации MSDN, зачастую стоит потратить время на испытание и массива, и коллекции, чтобы выбрать наиболее практичный и эффективный вариант.

В этом разделе решим две типичные задачи. Первая состоит в следующем: мы имеем список сотрудников предприятия, в котором есть следующие поля: имя сотрудника, его возраст и отметка, курит ли он. Из этого списка следует выбрать только некурящих сотрудников для повышения им зарплаты. Кроме того, из исходного списка выберем также сотрудников, чей возраст превышает 33 года («возраст Христа»). Таким образом, мы должны получить два новых списка и вывести на консоль фамилии из обоих списков.

Для решения этой задачи запустим Visual Studio 2010, далее создадим новый проект, в узле Visual C++ в среде CLR выберем шаблон Console Application CLR, укажем имя Name — LinqСписок1. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.2 приведем программный код данного приложения.

Листинг 11.2. Извлечение данных из списка (вариант 1)

```
// LinqСписок1.cpp: главный файл проекта.
// Программа из списка сотрудников некоторого предприятия выбирает
// в отдельный список только некурящих сотрудников
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим эти пространства имен для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
// Объявляем структуру (или класс):
value struct Сотрудник
// или value class Сотрудник
{
public: String ^ Имя;
public: int Возраст;
```

продолжение ⇨

Листинг 11.2 (продолжение)

```
public: bool КуритЛи;
};
bool Предикат(Сотрудник S)
{
    // Если сотрудник не курит, то заносим его в список некурящих:
    bool A = false;
    if (S.КуритЛи == false) A = true;
    return A;
    // Можно было бы записать короче:
    // return !S.КуритЛи;
    // но более запутанно
}
bool Предикат2(Сотрудник S)
{
    // Если сотруднику больше 33 лет, то заносим его в список "взрослых":
    bool A = false;
    if (S.Возраст > 33) A = true;
    return A;
    // Можно было бы записать короче:
    // return S.Возраст > 33;
    // но более запутанно
}
int main(array<System::String ^> ^args)
{
    Console::Title = "Фильтрация списка методами класса Linq::Enumerable";
    Console::BackgroundColor = ConsoleColor::Cyan;  // - цвет фона
    Console::ForegroundColor = ConsoleColor::Black; // - цвет текста
    Console::Clear();
    // Создаем список сотрудников:
    List<Сотрудник> ^ Сотрудники = gcnew List<Сотрудник>();
    // Инициализация списка сотрудников (их для упрощения всего четыре):
    Сотрудник Сотрудник1 = {"Зиборов Виктор", 45, false};
    Сотрудники->Add(Сотрудник1);
              Сотрудник Сотрудник2 = {"Еременко Татьяна", 22, true};
              Сотрудники->Add(Сотрудник2);
    Сотрудник Сотрудник3 = {"Стороженко Светлана", 32, false};
    Сотрудники->Add(Сотрудник3);
              Сотрудник Сотрудник4 = {"Тимощук Александр", 43, true};
              Сотрудники->Add(Сотрудник4);
    // Из списка сотрудников получаем новый список некурящих сотрудников:
    auto Некурящие = Enumerable::Where<Сотрудник>(Сотрудники, gcnew
                                          Func<Сотрудник ,bool>(Предикат));
    // Выводим полученный список на консоль:
    Console::WriteLine("Некурящие сотрудники:\n");
    for each (Сотрудник x in Некурящие)
        Console::WriteLine("{0} ", x.Имя);
    // Из списка сотрудников получаем новый список "взрослых" сотрудников,
    // возраст которых превышает 33 года:
```

```
    auto Взрослые = Enumerable::Where<Сотрудник>(Сотрудники, gcnew
                                        Func<Сотрудник ,bool>(Предикат2));
    // Выводим полученный список на консоль:
    Console::WriteLine("\n\"Взрослые\" сотрудники:\n");
    for each (Сотрудник x in Взрослые)
        Console::WriteLine("{0} ", x.Имя);
    Console::ReadKey();
    return 0;
}
```

Можно заметить, что значительная часть программного кода выполняет инициализацию списка сотрудников предприятия. Для упрощения кода мы рассматриваем список из четырех сотрудников. Этот фрагмент программы мы представили в «шахматном» порядке для большей структурированности программного кода. При создании списка объявлена структура Сотрудник, которая содержит три поля: Имя, Возраст и булеву переменную КуритЛи. Как и для случая манипуляций с массивом, для фильтрации исходного списка данных используем метод Where класса Enumerable. При этом условие выбора задаем в отдельной процедуре Предикат. В результате получаем новый список «Некурящие». Полученный список выводим на консоль, используя цикл for each (рис. 11.2). Аналогично получаем второй список «Взрослые», здесь условие выбора для этого списка задаем в другой процедуре Предикат2.

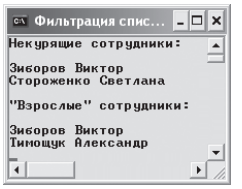

Рис. 11.2. Выборка данных из списка с помощью технологии LINQ

Вторая задача немного сложнее: требуется создать список студентов факультета, содержащий фамилию студента и массив полученных им текущих оценок, то есть массив оценок должен быть «вложен» в список студентов. Из списка студентов необходимо выбрать тех, кто имеет в перечне своих оценок хотя бы одну двойку, а также упорядочить полученный список в алфавитном порядке по фамилиям.

Как и в предыдущем случае, запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqСписок2. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core.

В листинге 11.3 приведен программный код данного приложения.

Листинг 11.3. Извлечение данных из списка (вариант 2)

```
// LinqСписок.cpp: главный файл проекта.
// Имеем список студентов с их фамилиями и текущими оценками. Программа
// фильтрует этот список для получения нового списка студентов, у которых
// среди текущих оценок имеется хотя бы одна двойка
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
```

продолжение ⇙

Листинг 11.3 (продолжение)

```
// Добавим для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
// Объявляем класс:
value class Студент
    // или структуру: value struct Студент
{
public: String ^ Фамилия;
public: array<int> ^ Оценки;
};
bool Предикат(Студент S)
{
    bool A = false;
    // Если хотя бы одна оценка - двойка, то выход из цикла,
    // а студента объявляем двоечником: A = true:
    for each(int i in S.Оценки)
        if (i <= 2) { A = true; break; }
        return A;
}
String ^ Предикат2(Студент S)
{
    // Сортировка в алфавитном порядке по фамилии:
    return S.Фамилия;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем массив объектов типа Студент:
    array<Студент> ^ Массив =
    { "Зиборов",     gcnew array<int>{5, 4, 4, 5},
      "Стороженко",  gcnew array<int>{3, 3, 2, 4},
      "Ломачинская", gcnew array<int>{3, 4, 4, 5},
      "Погребицкий", gcnew array<int>{2, 4, 3, 2},
      "Тимощук",     gcnew array<int>{2, 3, 4, 3},
    };
    // Создаем список студентов из массива:
    List<Студент> ^ Студенты = Enumerable::ToList<Студент>(Массив);
    // Запрос на студентов-двоечников:
    auto Двоечники = Enumerable::Where<Студент>(Студенты, gcnew
                                    Func<Студент ,bool>(Предикат));
    // Сортируем полученный список студентов в алфавитном порядке:
    Двоечники = Enumerable::OrderBy(Двоечники, gcnew
                                Func<Студент, String ^>(Предикат2));
    // Вывод результата запроса на консоль:
    for each (Студент ^ x in Двоечники)
        Console::WriteLine("{0} ", x->Фамилия);
    Console::ReadKey();
    return 0;
}
```

Как видно, в начале программы объявляем класс (можно структуру) Студент, который имеет поля: фамилию студента и массив оценок. Далее заполняем вспомогательный массив, из которого создаем список студентов. Затем строим запрос методами технологии LINQ сначала на выявление из списка студентов, имеющих неудовлетворительные оценки (метод Where), а затем сортируем их по фамилии в алфавитном порядке (метод OrderBy). Результат работы программы приводим на рис. 11.3.

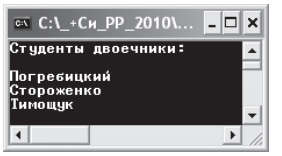

Рис. 11.3. Выборка двоечников из списка студентов

Убедиться в работоспособности этих двух программ можно, открыв соответствующие решения в папках LinqСписок1 и LinqСписок2.

Пример 91. Группировка данных методом GroupBy

Приведем примеры группировки данных в соответствии с заданной функцией селектора ключа методом GroupBy. В первом примере будем группировать данные в списке типа List. Каждая запись в этом списке представляет собой два поля: название месяца в году и количество дней в этом месяце. В этой задаче требуется получить два производных списка, в первый список будут входить месяцы, количество дней в которых равно 31, а во второй — остальные месяцы.

Как и в предыдущем случае, для решения этой задачи запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqМесяцы. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET дважды щелкнем на ссылке System.Core. В листинге 11.4 приведен программный код данного приложения.

Листинг 11.4. Группировка элементов списка данных методом GroupBy

```
// Linq_Месяцы.cpp: главный файл проекта.
// Программа создает список месяцев в году с указанием их названия
// и количеством дней в месяце. Затем создает запрос на группировку этих
// данных, то есть следует создать одну группу месяцев, в которых
// содержится 31 день, и другую группу, в которую входят прочие месяцы.
// Результат этого запроса программа выводит на консоль
#include "stdafx.h"
// Добавим на вкладке .NET ссылку на System.Core
using namespace System;
// Добавим для краткости выражений:
using namespace System::Linq;
using namespace System::Collections::Generic;
value struct Месяц
```

продолжение ⇩

Листинг 11.4 (продолжение)

```
{
public: String ^ Название;
public: int Дней;
};
bool Предикат(Месяц t)
{
    bool A = false;
    if (t.Дней == 31) A = true;
    return A;
}
Месяц Предикат2(Месяц t)
{
    return t;
}
int main(array<System::String ^> ^args)
{
    // Задаем цвет текста на консоли для большей выразительности:
    Console::ForegroundColor = ConsoleColor::White;
    // Создаем список месяцев - их 12:
    auto Месяцы = gcnew List<Месяц>(12);
    auto Месяц1 = Месяц();
    // Инициализация списка:
    Месяц1.Название = "Январь";   Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Февраль";  Месяц1.Дней = 28; Месяцы->Add(Месяц1);
    Месяц1.Название = "Март";     Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Апрель";   Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Май";      Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Июнь";     Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Июль";     Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Август";   Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Сентябрь"; Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Октябрь";  Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    Месяц1.Название = "Ноябрь";   Месяц1.Дней = 30; Месяцы->Add(Месяц1);
    Месяц1.Название = "Декабрь";  Месяц1.Дней = 31; Месяцы->Add(Месяц1);
    // Запрос на группировку данных, записанных в список Месяцы. Этот список
    // делим на две группы: одна группа включает в себя месяцы, содержащие
    // 31 день, вторая - прочие месяцы:
    auto Запрос = Enumerable::GroupBy<Месяц, bool, Месяц>(Месяцы,
                          gcnew Func<Месяц, bool>(Предикат),
                          gcnew Func<Месяц, Месяц>(Предикат2));
    // Выводим результаты запроса на консоль:
    Console::WriteLine("Две группы месяцев: \n");
    // Цикл по группам:
    for each (IGrouping<bool,Месяц> ^ Группа in Запрос)
    {
        if(Группа->Key == true)
            Console::WriteLine("Месяцы, содержащие 31 день: \n");
        else Console::WriteLine("\nПрочие месяцы: \n");
```

```
        // Цикл по месяцам в группе:
        for each (Месяц M in Группа)
            Console::WriteLine("{0} - {1}", M.Название, M.Дней);
    }
    // Ждем от пользователя нажатия какой-либо клавиши:
    Console::ReadKey();
    return 0;
}
```

В программном коде мы вначале создаем список типа List. В угловых скобках указываем, что каждая запись в списке будет представлять собой структуру Месяц, состоящую из двух полей строкового и целого типов. После инициализации списка организуем запрос на получение двух производных групп методом GroupBy. Результат запроса попадает в переменную Запрос. Для вывода обеих групп на консоль используем два вложенных цикла for each. Внешний цикл выполняется по группам, а второй выводит непосредственно название каждого месяца и соответственное количество дней на консоль. Результат работы программы представлен на рис. 11.4.

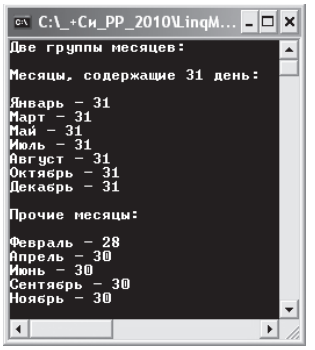

Рис. 11.4. Группировка списка по количеству дней в месяцах

Вторая задача, которую мы решим в данном разделе, манипулирует со словарем данных типа Dictionary. В словаре данных будем записывать информацию о некоторых товарах. В реальной ситуации данные о каждом товаре займут большое количество полей, то есть колонок в соответствующих таблицах. В нашем примере каждый товар будет иметь только название и цену. Чтобы ориентироваться в большом множестве товаров, это множество разбивают на группы. В нашей задаче мы наши товары, записанные в словарь данных, разделим на две ценовые группы, соответственно дороже и дешевле 90 рублей. Это деление выполним, используя LINQ-технологию.

Как и в предыдущем случае, запускаем Visual Studio 2010, далее создаем новый проект, в узле Visual C++ в среде CLR выбираем шаблон Console Application CLR, указываем имя Name — LinqЦеныНаПродукты. Далее, чтобы иметь доступ к пространству имен System::Linq, добавим в текущий проект объектную библиотеку System.Core.dll. Для этого выберем пункты меню Project ▸ Properties ▸ Add Reference и на вкладке .NET