**Вопросы 11 лабораторной LINQ to Object**

**1. Что такое LINQ?**

**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.

**2. В чем разница между отложенными операциями и не отложенными**

**операциями LINQ to Object?**

При отложенном выполнении LINQ-выражение не выполняется, пока не будет произведена итерация или перебор по выборке. С помощью ряда методов мы можем применить не отложенное выполнение запроса. Это методы, которые возвращают одно атомарное значение или один элемент. Например, Count(), Average(), First() / FirstOrDefault(), Min(), Max() и т.д.

**3. Что такое лямбда-выражения?**

Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы.

Ламбда-выражения имеют следующий синтаксис: слева от лямбда-оператора => определяется список параметров, а справа блок выражений, использующий эти параметры: (список\_параметров) => выражение.

**4. Как используется операция Where в LINQ to Object?**

Операция Where используется для фильтрации элементов в последовательность. Операция имеет два прототипа.

|  |  |
| --- | --- |
| *****Первый прототип Where***** | *****Второй прототип Where***** |
| Этот прототип Where принимает входную последовательность и делегат метода-предиката, а возвращает объект, который при перечислении проходит по входной последовательности, выдавая элементы, для которых делегат метода-предиката возвращает true. | Второй прототип Where идентичен первому, но с тем отличием, что он указывает на то, что делегат метода-предиката принимает дополнительный целочисленный аргумент. Этот аргумент будет индексом элемента во входной последовательности. |

**5. Как используется операция Select ?**

**Операция Select** используется для создания выходной последовательности одного типа элементов из входной последовательности элементов другого типа. Эти типы не обязательно должны совпадать.

|  |  |
| --- | --- |
| *Первый прототип Select* | *****Второй прототип Select***** |
| Этот прототип Select принимает входную последовательность и делегат метода-селектора в качестве входных параметров, а возвращает объект, который при перечислении проходит по входной последовательности и выдает последовательность элементов типа S. Как упоминалось ранее, T и S могут быть как одного, так и разных типов. | В этом прототипе операции Select методу-селектору передается дополнительный целочисленный параметр. Это индекс, начинающийся с нуля, входного элемента во входной последовательности. |

**6. Как используются операции Take, Skip?**

Операция Take возвращает указанное количество элементов из входной последовательности, начиная с ее начала. Операция Take имеет один прототип, описанный ниже:

public static IEnumerable<T> Take<T>(

this IEnumerable<T> source,

int count);

Этот прототип указывает, что Take принимает входную последовательность и целое число count, задающее количество элементов, которые нужно вернуть, и возвращает объект, который при перечислении выдает первые count элементов из входной последовательности.

**7. Как используется операция Concat ?**

Операция Concat соединяет две входные последовательности и выдает одну выходную последовательность. Операция имеет один прототип.

public static IEnumerable<T> Concat<T>(

this IEnumerable<T> first,

IEnumerable<T> second);

В этом прототипе две последовательности одного типа T — first и second — являются входными. Возвращается объект, который при перечислении проходит по первой последовательности, выдавая каждый ее элемент в выходную последовательности за которым начинается перечисление второй входной последовательности с выдачей каждого ее элемента в ту же выходную последовательность.

**8. Как используется операция OrderBy?**

Операция OrderBy позволяет упорядочить входную последовательность на основе метода keySelector, который возвращает значение ключа для каждого входного элемента. Упорядоченная выходная последовательность IOrderedEnumerable<T> выдается в порядке возрастания на основе значений возвращенных ключей.

|  |  |
| --- | --- |
| *Первый прототип OrderBy* | *****Второй прототип OrderBy***** |
| public static IOrderedEnumerable<T> OrderBy<T, K>(  this IEnumerable<T> source,  Func<T, K> keySelector)  where  К : IComparable<K>; | public static IOrderedEnumerable<T> OrderBy<T, K>(  this IEnumerable<T> source,  Func<T, K> keySelector,  IComparer<K> comparer); |
| В этом прототипе операции OrderBy передается входная последовательность source и делегат метода keySelector, а возвращается объект, который при перечислении проходит входную коллекцию source, собирая все элементы и передавая каждый из них методу keySelector, таким образом, извлекая каждый ключ и упорядочивая последовательность на основе этих ключей. | Этот прототип такой же, как первый, за исключением того, что он позволяет передавать объект-компаратор. Если используется эта версия операции OrderBy, то нет необходимости в том, чтобы тип K реализовывал интерфейс IComparable. |

**9. Как используется операция Join?**

Операция Join выполняет внутреннее соединение по эквивалентности двух последовательностей на основе ключей, извлеченных из каждого элемента этих последовательностей.

public static IEnumerable<V> Join<T, U, K, V>(

this IEnumerable<T> outer,

IEnumerable<U> inner,

Func<T, K> outerKeySelector,

Func<U, K> innerKeySelector,

Func<T, U, V> resultSelector);

**10. Как используются операции Distinct, Union, Except и Intersect?**

**Distinct**

Операция Distinct удаляет дублированные элементы из входной последовательности.

public static IEnumerable<T> Distinct<T>(

this IEnumerable<T> source);

Эта операция возвращает объект, перечисляющий элементы входной последовательности source и выдающий последовательность, в которой каждый элемент не эквивалентен предыдущим выданным. Эквивалентность элементов определяется методами GetHashCode и Equals.

**Union**

Операция Union возвращает объединение множеств из двух исходных последовательностей.

public static IEnumerable<T> Union<T>(

this IEnumerable<T> first,

IEnumerable<T> second);

Эта операция возвращает объект, который сначала перечисляет элементы последовательности по имени first, выдавая последовательность, в которой каждый элемент не эквивалентен предыдущим выданным, затем перечисляет вторую входную последовательность second, опять-таки, выдавая последовательность без повторений. Эквивалентность элементов определяется методами GetHashCode и Equals.

**Except**

Операция Except возвращает последовательность, содержащую все элементы первой последовательности, которых нет во второй последовательности.

public static IEnumerable<T> Except<T>(

this IEnumerable<T> first,

IEnumerable<T> second);

Эта операция возвращает объект, который при перечислении перебирает элементы входной последовательности по имени second, собирая все элементы, которые не эквивалентны ранее собранным. Затем происходит перечисление входной последовательности first, с выдачей каждого ее элемента, которого нет в коллекции из второй последовательности. Эквивалентность одного элемента другому определяется с использованием их методов GetHashCode и Equals.

**Intersect**

Операция Intersect возвращает пересечение множеств из двух исходных последовательностей.

public static IEnumerable<T> Intersect<T> (

this IEnumerable<T> first,

IEnumerable<T> second);

Эта операция возвращает объект, который сначала перечисляет элементы последовательности по имени first, выбирая оттуда каждый элемент, который не эквивалентен предыдущему выбранному элементу. Затем он перечисляет вторую входную последовательность, помечая любой элемент, имеющийся в обеих последовательностях, для включения в выходную последовательность. Затем осуществляется проход по помеченным элементам, с помещением их в выходную последовательность в том порядке, в котором они были собраны. Эквивалентность элементов определяется с помощью методов GetHashCode и Equals.

**11. Как используются операции First, Last, Any, All и Contains?**

**First**

Операция First возвращает первый элемент последовательности или первый элемент последовательности, соответствующий предикату — в зависимости от использованного прототипа.

public static T First<T>(

this IEnumerable<T> source);

При использовании этого прототипа операции First выполняется перечисление входной последовательности source и возвращается ее первый элемент.

Второй прототип операции First позволяет передать ему предикат:

public static T First<T>(

this IEnumerable<T> source,

Func<T, bool> predicate);

Эта версия операции First возвращает первый найденный ею элемент, для которого predicate дал true. Если ни один из элементов не заставил predicate вернуть true, то операция First генерирует исключение InvalidOperationException.

**Last**

Операция Last возвращает последний элемент последовательности или последний элемент, соответствующий предикату — в зависимости от используемого прототипа.

public static T Last<T>(

this IEnumerable<T> source);

В случае этого прототипа операция Last перечисляет входную последовательность по имени source и возвращает ее последний элемент.

public static T Last<T>(

this IEnumerable<T> source,

Func<T, bool> predicate) ;

Второй прототип Last позволяет передать predicate. Эта версия операции Last возвращает последний из найденных элементов, для которых predicate вернет true.

**Any**

Операция Any возвращает true, если любой из элементов входной последовательности отвечает условию.

public static bool Any<T> (

this IEnumerable<T> source);

Этот прототип операции Any вернет true, если входная последовательность source содержит любые элементы.

Второй прототип операции Any перечисляет входную последовательность и возвращает true, если хотя бы для одного элемента из входной последовательности вызов делегата метода predicate возвращает true. Перечисление входной последовательности source прекращается, как только predicate вернет true:

public static bool Any<T> (

this IEnumerable<T> source,

Func<T, bool> predicate);

**All**

Операция All возвращает true, если каждый элемент входной последовательности отвечает условию.

public static bool All<T>(

this IEnumerable<T> source,

Func<T, bool> predicate);

Операция All перечисляет входную последовательность source и возвращает true, только если predicate возвращает true для каждого элемента последовательности. После возврата из predicate значения false перечисление прекращается.

**Contains**

Операция Contains возвращает true, если любой элемент входной последовательности соответствует указанному значению.

public static bool Contains<T>(

this IEnumerable<T> source,

T value);

Этот прототип операции Contains сначала проверяет входную последовательность на предмет реализации ею **интерфейса ICollection<T>**, и если она его реализует, то вызывается метод Contains реализации последовательности. Если последовательность не реализует интерфейс ICollection<T>, входная последовательность source перечисляется с проверкой соответствия каждого элемента указанному значению. Как только найден элемент, который ему соответствует, перечисление прекращается.

Второй прототип подобен первому, за исключением возможности указания объекта IEqualityComparer<T>. Если используется этот прототип, каждый элемент в последовательности сравнивается с переданным значением с применением переданного объекта проверки эквивалентности:

public static bool Contains<T>(

this IEnumerable<T> source,

T value,

IEqualityCoinparer<T> comparer);

**12. Как используются операции Count, Sum, Min и Max, Average?**

**Count**

**Sum**

Метод Sum() имеет ряд перегрузок. В частности, если у нас набор сложных объектов, то мы можем указать свойство, значения которого будут суммироваться: users.Sum(n => n.Age)

**Min**

**Max**

**Average**

Операция Average возвращает среднее арифметическое числовых значений элементов входной последовательности.

public static Result Average(

this IEnumerable<Numeric> source);

Тип Numeric должен быть одним из int, long, double или decimal, либо одним из их допускающих null эквивалентов: int?, long?, double? или decimal?. Если тип Numeric — int или long, то типом Result будет double. Если же тип Numeric — int? или long?, то типом Result будет double?. В противном случае тип Result совпадает с типом Numeric. Первый прототип операции Average перечисляет входную последовательность source элементов типа Numeric, вычисляя их среднее значение.

Второй прототип операции Average перечисляет входную последовательность source элементов и определяет среднее значение членов элементов, возвращаемых функцией selector для каждого элемента входной последовательности.

public static Result Average<T>(

this IEnumerable<T> source,

Func<T, Numeric> selector);

Если любой из аргументов равен null, генерируется исключение ArgumentNullException. Если сумма усредняемых значений превышает емкость long для типов Numeric — int, int?, long и long?, генерируется исключение OverflowException.