

Операции Cast, OfType и AsEnumerable

[LINQ \(../base/level1/info_linq.php\)](#) --- [LINQ to Objects \(../level1/linq_index.php\)](#) --- [Операции Cast, OfType и AsEnumerable](#)

Операции преобразования предоставляют простой и удобный способ преобразования последовательностей в другие типы коллекций.

Cast

Операция Cast используется для приведения каждого элемента входной последовательности в выходную последовательность указанного типа. Операция Cast имеет один прототип, описанный ниже:

C#

```
public static IEnumerable<T> Cast<T>(
    this IEnumerable source);
```

Первое, что следует отметить в операции Cast — это то, что ее первый аргумент по имени source имеет тип IEnumerable, а не IEnumerable<T>, тогда как большинство отложенных стандартных операций запросов принимают первый аргумент типа IEnumerable<T>. Это связано с тем, что операция Cast предназначена для вызова на классах, реализующих интерфейс IEnumerable, а не IEnumerable<T>. В частности, речь идет об унаследованных коллекциях, разработанных до появления версии C# 2.0 и обобщений.

Операцию Cast можете вызывать на унаследованных коллекциях C# до тех пор, пока они реализуют IEnumerable, и при этом будет создана выходная последовательность IEnumerable<T>. Поскольку большинство стандартных операций запросов работает на последовательностях типа IEnumerable<T>, для преобразования к ним унаследованной коллекции понадобится вызвать метод вроде этого или, возможно, операцию OfType, которая рассматривается ниже.

Пройди тесты

Эта операция вернет объект, который при перечислении проходит по исходной коллекции данных, преобразуя каждый элемент к типу T. Если элемент не может быть приведен к типу T, генерируется исключение. Из-за этого данная операция должна

[C# тест \(легкий\) \(https://professorweb.ru/test/c-sharp-test.html\)](https://professorweb.ru/test/c-sharp-test.html)

[.NET тест \(средний\) \(https://professorweb.ru/test/asp-test.html\)](https://professorweb.ru/test/asp-test.html)

вызываться, только когда точно известно, что каждый элемент входной последовательности может быть преобразован в тип T.

Пытаясь выполнять запросы LINQ на унаследованных коллекциях, не забудьте вызывать на них операцию Cast или OfType, чтобы создать последовательность IEnumerable<T>, на которой затем можно вызывать стандартные операции запросов.

В приведенном ниже примере с помощью метода GetEmployeesArrayList общего класса Employee (2_0.php) будет возвращена унаследованная, необобщенная коллекция ArrayList. Ниже содержится код, демонстрирующий, как тип данных элементов ArrayList может быть приведен к элементам последовательности IEnumerable<T>:

C#

```
ArrayList employees = Employee.GetEmployeesArrayList();
Console.WriteLine("Тип данных employees: " + employees.GetType());

var seq = employees.Cast<Employee>();
Console.WriteLine("Тип данных seq: " + seq.GetType());

var emps = seq.OrderBy(e => e.firstName);
foreach (Employee emp in emps)
    Console.WriteLine("{0} {1}", emp.firstName, emp.lastName);
```

Сначала производится вызов метода GetEmployeesArrayList, который вернет коллекцию ArrayList объектов Employee и затем отображается тип данных переменной employees. Далее этот ArrayList преобразуется в последовательность IEnumerable<T>, вызывая для этого операцию Cast, после чего отображается тип данных возвращенной последовательности. И, наконец, для демонстрации, что все работает, элементы возвращенной последовательности выводятся на консоль. Ниже показан результат выполнения этого кода:



Пройди тесты

Как видите, типом данных переменной employees является ArrayList. Немного сложнее выяснить тип данных seq. Определенно можно сказать, что он отличается и выглядит как последовательность. Кроме того, в наименовании его типа присутствует

слово `CastIterator`. Вы должны помнить, что отложенные операции не возвращают немедленно выходную последовательность, а возвращают объект, перечисление по которому будет выдавать элементы в выходную последовательность. Тип данных переменной `seq` в предыдущем примере указывает именно на объект такого рода. Тем не менее, это деталь реализации, которая в будущем может измениться.

Операция `Cast` попытается привести каждый элемент входной последовательности к указанному типу. Если любой из этих элементов не может быть приведен к указанному типу, будет сгенерировано **исключение `InvalidCastException`**. Если существует вероятность присутствия разнотипных элементов в исходной коллекции, применяйте вместо `Cast` операцию `OfType`.

OfType

Операция `OfType` используется для построения выходной последовательности, содержащей только те элементы, которые могут быть успешно преобразованы к указанному типу.

Эта операция имеет один прототип, описанный ниже:

C#

```
public static IEnumerable<T> OfType<T>(
    this IEnumerable source);
```

Первое, что следует отметить в операции `OfType` — это то, что ее первый аргумент по имени `source`, как и у операции `Cast`, имеет тип `IEnumerable`, а не `IEnumerable<T>`. Большинство первых аргументов отложенных стандартных операций запросов имеет тип `IEnumerable<T>`. Это объясняется тем, что операция `OfType` предназначена для вызова на классах, реализующих интерфейс `IEnumerable`, а не `IEnumerable<T>`.

Таким образом, вызывать операцию `OfType` на унаследованной коллекции можно до тех пор, пока она реализует `IEnumerable`, и при этом будет создана выходная последовательность `IEnumerable<T>`. Поскольку большинство стандартных операций запросов работает на последовательностях типа `IEnumerable<T>`, для преобразования к ним унаследованной коллекции понадобится вызвать метод вроде этого или, возможно, операцию `OfType`. Это важно, когда вы пытаетесь использовать стандартные операции запросов на унаследованных коллекциях.

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

Операция OfType вернет объект, который при перечислении проходит по входной последовательности, выдавая только элементы, тип которых преобразуется к указанному типу T.

Операция OfType отличается от Cast тем, что Cast пытается привести каждый элемент входной последовательности к типу T и выдать его в выходную последовательность. Если приведение не удастся, генерируется исключение. Операция OfType пытается выдать входной элемент только в том случае, если он может быть приведен к типу T. Формально чтобы элемент e попал в выходную последовательность, для него должно быть истинно выражение `e is T`.

Для примера ниже создается коллекция ArrayList, содержащая объекты двух общих классов — Employee и EmployeeOptionEntry (2_0.php). После наполнения ArrayList объектами обоих классов сначала вызывается операция Cast, чтобы продемонстрировать, что она терпит неудачу в данной ситуации. Затем выполняется вызов операции OfType:

C#

```
ArrayList al = new ArrayList();
    al.Add(new Employee { id = 1, firstName = "Joe", lastName = "Rattz" });
    al.Add(new Employee { id = 2, firstName = "William", lastName = "Gates" });
    al.Add(new EmployeeOptionEntry { id = 1, optionsCount = 0 });
    al.Add(new EmployeeOptionEntry { id = 2, optionsCount = 9999999999 });
    al.Add(new Employee { id = 3, firstName = "Anders", lastName = "Hejlsberg" });
    al.Add(new EmployeeOptionEntry { id = 3, optionsCount = 848475745 });

    var items = al.Cast<Employee>();

    Console.WriteLine("Попытка использования операции Cast...");
    try
    {
        foreach (Employee item in items)
            Console.WriteLine("{0} {1} {2}", item.id, item.firstName, item.lastName);
    }
    catch (Exception ex)
    {
        Console.WriteLine("{0}{1}", ex.Message, System.Environment.NewLine);
    }

    Console.WriteLine("Попытка использования операции OfType...");
    var items2 = al.Of<Employee>();
```

Пройди тесты

```
foreach (Employee item in items2)
    Console.WriteLine("{0} {1} {2}", item.id, item.firstName, item.lastName);
```

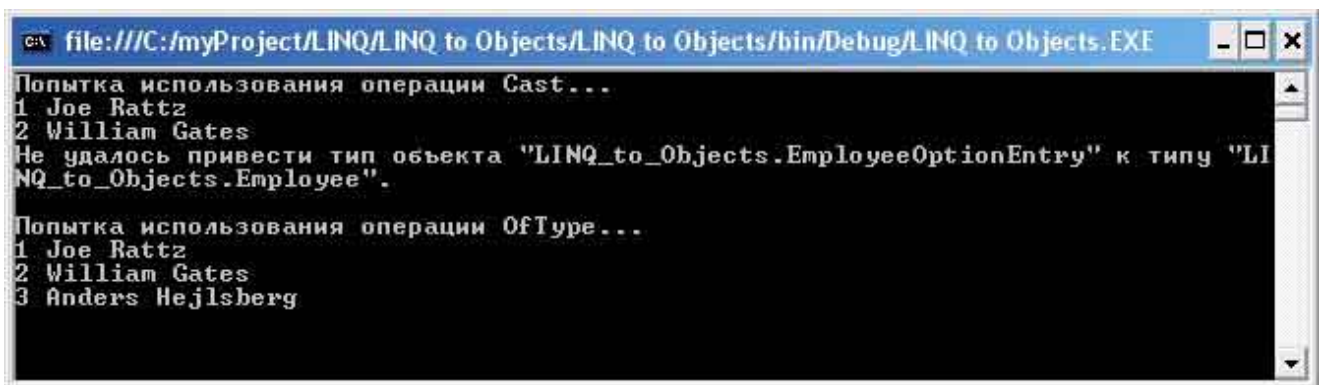
C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

После создания и наполнения коллекции `ArrayList` элементами вызывается операция `Cast`. Следующий шаг — попытка перечисления результатов. Этот шаг необходим, поскольку операция `Cast` является отложенной. Если никогда не перечислять результаты этого запроса, он никогда не будет выполнен и проблема не обнаружится.

Обратите внимание, что цикл `foreach`, который перечисляет результаты запроса, заключен в блок `try/catch`. В данном случае это необходимо, поскольку известно, что возникнет исключение по причине наличия объектов двух совершенно разных типов. Затем вызывается операция `OfType` и выполняется перечисление с отображением его результатов. Обратите внимание, что этот цикл `foreach` не помещен в блок `try/catch`. Разумеется, в реальном рабочем коде не следует игнорировать защиту, которую предоставляет блок `try/catch`.

Ниже показан результат выполнения этого запроса:



```
file:///C:/myProject/LINQ/LINQ to Objects/LINQ to Objects/bin/Debug/LINQ to Objects.EXE
Попытка использования операции Cast...
1 Joe Rattz
2 William Gates
Не удалось привести тип объекта "LINQ_to_Objects.EmployeeOptionEntry" к типу "LINQ_to_Objects.Employee".
Попытка использования операции OfType...
1 Joe Rattz
2 William Gates
3 Anders Hejlsberg
```

Обратите внимание, что полностью выполнить перечисление результата запроса операции `Cast` без генерации исключения не удалось. Но все прошло нормально при перечислении результата операции `OfType`; в этом случае в выходную последовательность были включены только элементы типа `Employee`.

Мораль сей басни в том, что если входная последовательность может содержать элементы более чем одного типа данных, следует отдать предпочтение операции `OfType` перед `Cast`.

AsEnumerable

Операция `AsEnumerable` возвращает входную последовательность типа `IEnumerable<T>` как тип `IEnumerable<T>`. Операция `AsEnumerable` имеет один прототип, который описан ниже:

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

C# .NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)

```
public static IEnumerable<T> AsEnumerable<T>(
    this IEnumerable<T> source);
```

Приведенный выше прототип объявляет, что операция `AsEnumerable` работает над входной последовательностью типа `IEnumerable<T>` по имени `source` и возвращает ту же последовательность типа `IEnumerable<T>`. Он служит ни чему иному, кроме как изменению типа выходной последовательности во время компиляции.

Это может показаться излишним, поскольку операция уже вызывается на типе `IEnumerable<T>`. Может возникнуть вопрос, зачем нужно преобразовывать последовательность `IEnumerable<T>` в последовательность типа `IEnumerable<T>`? Ответ будет дан ниже.

Стандартные операции запросов объявлены для работы с нормальными последовательностями LINQ to Objects — коллекциями, реализующими интерфейс `IEnumerable<T>`. Однако другие специфичные для предметной области коллекции, такие как обращающиеся к базе данных, могут реализовывать собственные типы последовательностей со своими операциями. Обычно при вызове операции запроса на коллекции одного из таких типов должны вызываться операции, специфичные для данного типа. Операция `AsEnumerable` позволяет привести входную коллекцию к нормальному типу последовательности `IEnumerable<T>`, позволяя вызывать на ней методы стандартных операций запросов.

Например, когда речь пойдет о LINQ to SQL в другом разделе, вы увидите, что LINQ to SQL в действительности использует собственный тип последовательности — `IQueryable<T>` — и реализует собственные операции. Вызов метода `Where` на последовательности типа `IQueryable<T>` — это на самом деле вызов метода `Where` из LINQ to SQL, а не стандартная операция запроса `Where` из LINQ to Objects!

Попытка вызова одной из стандартных операций запросов приведет к исключению, если только не окажется одноименной операции LINQ to SQL. С помощью операции `AsEnumerable` можно выполнить приведение последовательности `IQueryable<T>` к последовательности `IEnumerable<T>`, что позволит вызывать на ней стандартные операции запросов. Это бывает очень удобно, когда нужно контролировать, в каком API-интерфейсе вызывается операция.

Предыдущая (2_9.php)	9	10	11	Вперед (2_11.php)
----------------------	---	----	----	-------------------

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)



Лучший чат для C# программистов (<https://t.me/professorweb>)

Professor Web (/)

Наш любимый хостинг (/)

Пройди тесты

C# тест (легкий) (<https://professorweb.ru/test/c-sharp-test.html>)

.NET тест (средний) (<https://professorweb.ru/test/asp-test.html>)