# **[LINQ标准查询操作符(四) —AsEnumerable,Cast,OfType,ToArray,ToDictionary,ToList,ToLookup,First,Last,ElementAt](http://www.cnblogs.com/dotnetmvc/p/3680481.html)**

# **十、转换操作符**

转换操作符是用来实现将输入对象的类型转变为序列的功能。名称以“As”开头的转换方法可更改源集合的静态类型但不枚举（延迟加载）此源集合。名称以“To”开头的方法可枚举（即时加载）源集合并将项放入相应的集合类型。

## **1. AsEnumerable**

所有实现了IEnumerable<T>接口的类型都可以调用此方法来获取一个IEnumerable<T>集合。此方法一般仅用于实现类中的方法与IEnumerable<T>接口方法重名时。例如，实现类Test中有一个Where方法，当使用Test对象调用Where时，将执行Test自身的Where方法过程。如果要执行IEnumerable<T>的Where方法，便可以使用AsEnumerable进行进行转换后，再调用Where方法即可。当然，将实现类Test隐式转换为IEnumerable<T>接口，再调用接口的Where方法也能达到同样的效果。以下的代码演示了这一过程：

[IMG_256](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

class AsEnumerableTest<T> : List<T>

{

public void Where(Func<T, bool> func)

{

Console.WriteLine("AsEnumerableTest的Where方法");

}

}public static void AsEnumerable()

{

AsEnumerableTest<int> q = new AsEnumerableTest<int>() { 1,2,3,4 };

q.Where(r => r < 3);

//q.AsEnumerable().Where(r => r < 3);

//IEnumerable<int> i = q;

//i.Where(r => r < 3);

}

[IMG_257](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

## **2. Cast**

Cast<T> 方法通过提供必要的类型信息，可在IEnumerable(非泛型)的派生对象上调用Cast<T> 方法来获得一个IEnumerable<T>对象。例如，ArrayList 并不实现 IEnumerable<T>，但通过调用 ArrayList 对象上的 Cast<T>()，就可以使用标准查询运算符查询该序列。

如果集合中的元素无法强制转换为 T 类型，则此方法将引发异常。以下代码演示了这一过程：

[IMG_258](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

ArrayList array = new ArrayList();

array.Add("Bob");

array.Add("Jack");

array.Add(1);

foreach (var item in array.Cast<string>())

{

Console.WriteLine(item);

}

[IMG_259](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

运行此代码，可以输出“Bob”、“Jack”，然后会报出一个异常“无法将int强制转换为string”，这说明Cast方法也是延迟执行实现的，只有在枚举过程中才将对象逐个强制转换为T类型。

## **3. OfType**

OfType <T> 方法通过提供必要的类型信息，可在IEnumerable(非泛型)的派生对象上调用OfType <T> 方法来获得一个IEnumerable<T>对象。执行OfType<T>方法将返回集合中强制转换类型成功的所有元素。也就是说，OfType<T>方法与Cast<T> 方法的区别在于，如果集合中的元素在强制转换失败的时候会跳过，而不是抛出异常。

## **4. ToArray**

ToArray 操作符可以在IEnumerable<T> 类型的任何派生对象上调用，返回值为T类型的数组。

## **5. ToDictionary**

ToDictionary操作符根据指定的键选择器函数，从IEnumerable<T>创建一个Dictionary<TKey, TValue>。下面的示例中，将查询到的产品类别集合转换为Dictionary<类别ID,类别名称>的键-值集合：

[IMG_260](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

var q =

db.Categories

.ToDictionary

(

c => c.CategoryID,

c => c.CategoryName

);

生成的sql:

SELECT

[Extent1].[CategoryID] AS [CategoryID],

[Extent1].[CategoryName] AS [CategoryName],

[Extent1].[Description] AS [Description],

[Extent1].[Picture] AS [Picture]

FROM [dbo].[Categories] AS [Extent1]

var qq =

db.Categories

.Select(c => new { c.CategoryID, c.CategoryName })

.ToDictionary(c => c.CategoryID, c => c.CategoryName);

生成的sql:

SELECT

[Extent1].[CategoryID] AS [CategoryID],

[Extent1].[CategoryName] AS [CategoryName]

FROM [dbo].[Categories] AS [Extent1]

[IMG_261](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

需要注意的是，如果省略ToDictionary方法的第二个参数（值选择函数），那么Value将会保存一个类别对象。还有，如果Key为null，或者出现重复的Key，都将导致抛出异常。

## **6. ToList**

ToList操作符可以在IEnumerable<T> 类型的任何派生对象上调用，返回值为List<T>类型的对象。

## **7. ToLookup**

ToLookup操作符将创建一个 Lookup<TKey, TElement>对象，这是一个one-to-many集合，一个Key可以对应多个Value。以下的示例以产品表的所有数据作为数据源，以类别ID作为Key调用了ToLookup方法，然后遍历返回的Lookup<TKey, TElement>对象，输出了类别ID以及此类别下的所有产品名称：

[IMG_262](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

var q =

db.Products

.ToLookup

(

p => p.CategoryID,

p => p.ProductName

);

生成的sql:

SELECT

[Extent1].[ProductID] AS [ProductID],

[Extent1].[ProductName] AS [ProductName],

[Extent1].[SupplierID] AS [SupplierID],

[Extent1].[CategoryID] AS [CategoryID],

[Extent1].[QuantityPerUnit] AS [QuantityPerUnit],

[Extent1].[UnitPrice] AS [UnitPrice],

[Extent1].[UnitsInStock] AS [UnitsInStock],

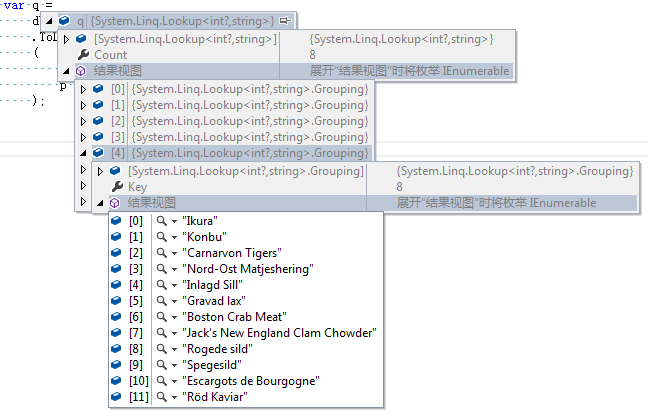
[Extent1].[UnitsOnOrder] AS [UnitsOnOrder],

[Extent1].[ReorderLevel] AS [ReorderLevel],

[Extent1].[Discontinued] AS [Discontinued]

FROM [dbo].[Products] AS [Extent1]

[IMG_263](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

返回的结果  


可以看出，ToLookup操作与GroupBy操作很相似，只不过GroupBy是延迟加载的，而ToLookup是即时加载。

如果省略第二个参数，则返回产品包装类。

# **十一、元素操作符**

元素操作符将从一个序列中返回单个指定的元素。

## **1. First**

First操作将返回序列中的第一个元素。如果序列中不包含任何元素，则First<T>方法将引发异常。若要在源序列为空时返回默认值，需要使用FirstOrDefault方法。以下代码演示了First<T>方法的使用方式：

[IMG_265](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

//无参

var query =

db.Employees

.First();

生成的sql:

SELECT TOP (1)

[c].[EmployeeID] AS [EmployeeID],

[c].[LastName] AS [LastName],

[c].[FirstName] AS [FirstName],

[c].[Title] AS [Title],

[c].[TitleOfCourtesy] AS [TitleOfCourtesy],

[c].[BirthDate] AS [BirthDate],

[c].[HireDate] AS [HireDate],

[c].[Address] AS [Address],

[c].[City] AS [City],

[c].[Region] AS [Region],

[c].[PostalCode] AS [PostalCode],

[c].[Country] AS [Country],

[c].[HomePhone] AS [HomePhone],

[c].[Extension] AS [Extension],

[c].[Photo] AS [Photo],

[c].[Notes] AS [Notes],

[c].[ReportsTo] AS [ReportsTo],

[c].[PhotoPath] AS [PhotoPath]

FROM [dbo].[Employees] AS [c]

//有参

var q =

db.Employees

.First(e => e.FirstName.StartsWith("S"));

生成的sql:

SELECT TOP (1)

[Extent1].[EmployeeID] AS [EmployeeID],

[Extent1].[LastName] AS [LastName],

[Extent1].[FirstName] AS [FirstName],

[Extent1].[Title] AS [Title],

[Extent1].[TitleOfCourtesy] AS [TitleOfCourtesy],

[Extent1].[BirthDate] AS [BirthDate],

[Extent1].[HireDate] AS [HireDate],

[Extent1].[Address] AS [Address],

[Extent1].[City] AS [City],

[Extent1].[Region] AS [Region],

[Extent1].[PostalCode] AS [PostalCode],

[Extent1].[Country] AS [Country],

[Extent1].[HomePhone] AS [HomePhone],

[Extent1].[Extension] AS [Extension],

[Extent1].[Photo] AS [Photo],

[Extent1].[Notes] AS [Notes],

[Extent1].[ReportsTo] AS [ReportsTo],

[Extent1].[PhotoPath] AS [PhotoPath]

FROM [dbo].[Employees] AS [Extent1]

WHERE [Extent1].[FirstName] LIKE N'S%'

[IMG_266](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

上述代码中使用了First<T>方法的无参方式与有参方式。First<T>的有参方式中可以指定一个条件，操作将返回序列中满足此条件的第一个元素。从查询结果上看，source.First<T>(条件)方法与source.Where(条件).First<T>()是一样的，但是需要注意“First<T>(条件)操作将返回序列中满足此条件的第一个元素”，这将忽略后面的遍历操作，效率更高。

## **2. FirstOrDefault**

FirstOrDefault方法将返回序列中的第一个元素；如果序列中不包含任何元素，则返回默认值。它也可以像First方法一样传递一个条件。需要说明的是如果序列中不包含任何元素，返回的默认值是个怎样的元素。在这之前，先来看一下FirstOrDefault方法是如何实现的：

[IMG_267](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

public static TSource FirstOrDefault<TSource>(this IEnumerable<TSource> source)

{

if (source == null)

{

throw Error.ArgumentNull("source");

}

IList<TSource> list = source as IList<TSource>;

if (list != null)

{

if (list.Count > 0)

{

return list[0];

}

}

else

{

using (IEnumerator<TSource> enumerator = source.GetEnumerator())

{

if (enumerator.MoveNext())

{

return enumerator.Current;

}

}

}

return default(TSource);

}

[IMG_268](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

1.         如果调用FirstOrDefault方法的序列为空，抛出异常

2.         如果序列成功转换为List<T>，并且元素数量大于0，则返回首个元素

3.         如果序列没有成功转换为List<T>，则尝试获取序列的遍历器，然后再调用遍历器的MoveNext方法，如果返回值为true，则返回当前的元素。

4.         如果上述操作都没有执行，则使用default(T)关键字返回类型T的默认值

以下给出MSDN中，对于default(T)关键字的描述：

在泛型类和泛型方法中产生的一个问题是，在预先未知以下情况时，如何将默认值分配给参数化类型 T：

* T 是引用类型还是值类型。
* 如果 T 为值类型，则它是数值还是结构。

给定参数化类型 T 的一个变量 t，只有当 T 为引用类型时，语句 t = null 才有效；只有当 T 为数值类型而不是结构时，语句 t = 0 才能正常使用。解决方案是使用**default** 关键字，此关键字对于引用类型会返回 null，对于数值类型会返回零。对于结构，此关键字将返回初始化为零或 null 的每个结构成员，具体取决于这些结构是值类型还是引用类型。

[IMG_269](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

//var q =

// db.Employees

// .First(e => e.FirstName.StartsWith("Svvvvvvvvv"));//没有记录报错

var qqq =

db.Employees

.FirstOrDefault(e => e.FirstName.StartsWith("Svvvvvvvvv"));//没有记录为 null

var qq =

db.Employees

.FirstOrDefault();

[IMG_270](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

## **3. Last**

Last方法将返回序列中的最后一个元素。使用方法参照First。

## **4. LastOrDefault**

LastOrDefault方法将返回序列中的最后一个元素；如果序列中不包含任何元素，则返回默认值。使用方法参照FirstOrDefault。

## **5. ElementAt**

ElementAt方法返回序列中指定索引处的元素。使用方法参照First。需要注意的是如果索引超出范围会导致异常。

## **6. ElementAtOrDefault**

ElementAtOrDefault方法将返回序列中指定索引处的元素；如果索引超出范围，则返回默认值。使用方法参照FirstOrDefault。

## **7. Single**

Single方法的无参形式将从一个序列中返回单个元素，如果该序列包含多个元素，或者没有元素数为0，则会引发异常。也就是说，在序列执行Single方法的无参形式时，必须保证该序列有且仅有一个元素。

Single方法的有参形式将从一个序列中返回符合指定条件的唯一元素，如果有多个元素，或者没有元素符合这一条件，则会引发异常。以下代码演示了Single的使用方式：

[IMG_271](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

//方法语法

//var q = //报错：序列包含一个以上的元素

// db.Employees

// .Single();

var query =

db.Employees

.Single(e => e.FirstName.StartsWith("S"));

[IMG_272](http://www.cnblogs.com/dotnetmvc/p/javascript:void(0);" \o "复制代码)

SingleOrDefault方法的无参形式将从一个序列中返回单个元素。如果元素数为0，则返回默认值。如果该序列包含多个元素，则会引发异常。

SingleOrDefault方法的有参形式将从一个序列中返回符合指定条件的唯一元素，如果元素数为0，则返回默认值；如果该序列包含多个元素，则会引发异常。SingleOrDefault的使用方式与Single相同。

需要注意的是，Single方法与SingleOrDefault方法都是即时加载的，在代码进行到方法所在位置时，如果引发了异常，会立刻抛出。

http://www.cnblogs.com/dotnetmvc/p/3680481.html