LINQ 进阶

回顾

- LINQ由哪几部分组成?
- LINQ是哪几个英文单词的缩写，具体是什么意思?
- 要使用 LINQ 特性，必须引入哪个命名空间?

本章目标

- 理解 LINQ 查询的执行时机
- 掌握查询语句
- 理解查询语句与查询方法的关系
- 掌握各种高级查询方法
- 理解 LINQ to SQL

查询执行的时机

请判断以下代码输出结果是什么？

```
int[] numbers = new int[]
   { 6, 4, 3, 2, 9, 1, 7, 8, 5 };

var even = numbers
   .Where(p => p % 2 == 0)
   .Select(p =>
   {
      Console.WriteLine("Hi! " + p.ToString());
      return p;
   });
```

演示示例：查询的时机

查询执行的时机

从前面的试验中，我们发现一次查询实际经过以下三步

1. 获取数据源

```
int[] numbers = new int[]
                { 6, 4, 3, 2, 9, 1, 7, 8, 5 };
```

2. 定义查询

```
var even = numbers
   .Where(p => p % 2 == 0)
   .Select(p =>
   {
      Console.WriteLine("Hi! " + p.ToString());
      return p;
   });
```

3. 执行查询

```
foreach (var item in even) { }
```

查询执行的时机小结

- 查询分为以下三步：获取数据源、定义查询、执行查询；
- 定义查询后，查询直到需要枚举结果时才被真正执行，这种方式称为“延迟执行(deferred execution)”；
- 当查询方法返回单一值时，查询立即执行；
- 因此，可以通过以下技巧在定义查询时就强制执行查询；

```
var even = numbers
   .Where(p => p % 2 == 0)
   .Select(p =>
   {
      Console.WriteLine("Hi! " + p.ToString());
      return p;
   }).Count();
```

LINQ查询的两种方式

事实上，LINQ查询存在以下两种形式

- Method Syntax, 查询方法方式
  - 主要利用 System.Linq.Enumerable 类中定义的扩展方法和 Lambda 表达式方式进行查询
  - 上一章的例子都是以这种方式查询
- Query Syntax, 查询语句方式
  - 一种更接近 SQL 语法的查询方式
  - 可读性更好

查询语句

```
int[] numbers = new int[] { 6, 4, 3, 2, 9, 1, 7, 8, 5 };

var even = from number in numbers
           where number % 2 == 0
           orderby number descending
           select number;
```

查询语句

两者的执行效果完全一样

```
int[] numbers = new int[] { 6, 4, 3, 2, 9, 1, 7, 8, 5 };

var even = numbers
    .Where(p => p % 2 == 0)
    .OrderByDescending(p => p)
    .Select(p => p);
```

查询方法

演示示例：查询语句与查询方法对比示例

更复杂的查询语句示例

请判断以下代码输出结果是什么？

```
List<Person> foxRiver8 = GetFoxRiver8();
var q = from p in foxRiver8
        where p.Age <= 30 && p.FirstName.Length == 7
        orderby p.Age descending
        select new{
           Name = p.FirstName + " " + p.LasName,
           Age = p.Age};

foreach (var item in q)
{
            Console.WriteLine(item.Name + " " + item.Age);
}
```

演示示例：复杂的查询语句示例

查询语句VS查询方法

查询语句与查询方法存在着紧密的关系

- CLR本身并不理解查询语句，它只理解查询方法
- 编译器负责在编译时将查询语句翻译为查询方法
- 大部分查询方法都有对应的查询语句形式：如 Select() 对应 select 、 OrderBy() 对应 orderby
- 部分查询方法目前在C#中还没有对应的查询语句：如 Count()和Max() 这时只能采用以下替代方案
  - 查询方法
  - 查询语句 + 查询方法的混合方式;
- 一般情况下，建议使用可读性更好的查询语句

高级查询方法

高级查询方法

- 聚合类
  - Count，Max/Min，Average
- 排序类
  - ThenBy
- 分区类
  - Take，TakeWhile，Skip，SkipWhile
- 集合类
  - Distinct
- 生成类
  - Range, Repeat

聚合类查询方法

聚合类查询方法

- Count
- Max/Min
- Average

Count 示例

Count 返回集合项的数目

```
int count = (from p in foxRiver8
             where p.Age <= 30
             select p).Count();
```

混合模式

```
int count = foxRiver8
            .Where(p => p.Age <= 30)
            .Count();
```

纯粹查询方法模式

演示示例：Count 方法示例

Max 示例

Max 返回集合中的最大值

```
int maxAge = (from p in foxRiver8
              select p.Age).Max();
```

混合模式

```
int maxAge = foxRiver8
             .Select(p => p.Age)
             .Max();
```

纯粹查询方法模式

演示示例：Max 方法示例

Min 示例

Min 返回集合中的最小值

```
int maxAge = (from p in foxRiver8
              select p.Age).Min();
```

混合模式

```
int maxAge = foxRiver8
             .Select(p => p.Age)
             .Min();
```

纯粹查询方法模式

演示示例：Min 方法示例

Average 示例

Average 返回集合的平均值

```
double averageAge = (from p in foxRiver8
                     select p.Age).Average();
```

混合模式

```
double averageAge = foxRiver8
                    .Select(p => p.Age)
                    .Average();
```

纯粹查询方法模式

演示示例：Average 方法示例

Sum 示例

Sum 返回集合的总和

```
Int sumAge = (from p in foxRiver8
              select p.Age).Sum();
```

混合模式

```
int sumAge = foxRiver8
             .Select(p => p.Age)
             .Sum();
```

纯粹查询方法模式

演示示例：Sum 方法示例

排序类查询方法

排序类查询方法

- ThenBy

ThenBy 示例

ThenBy 提供复合排序条件

```
var q = foxRiver8
        .OrderBy(p => p.FirstName)
        .ThenBy(p => p.LasName)
        .ThenBy(p => p.Age);
```

查询方法

```
var q = from p in foxRiver8
    orderby p.FirstName, p.LasName, p.Age
    select p;
```

查询语句

演示示例：ThenBy 方法示例

分区类查询方法

分区类查询方法

- Take/TakeWhile
- Skip/SkipWhile

Take/Skip 示例

- Take 提取指定数量的项
- Skip 跳过指定数量的项并获取剩余的项

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var q = numbers.Skip(1).Take(3);

foreach (var item in q)
{
   Console.WriteLine(item);
}
```

跳过前1条记录，连续提取3条记录，得到 2 3 4

演示示例：Skip/Take 方法示例

TakeWhile/SkipWhile 示例

TakeWhile 根据指定条件提取项

SkipWhile 根据指定条件跳过项

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var q = numbers.SkipWhile(i => i % 3 != 0)
   .TakeWhile(i => i % 2 != 0);

foreach (var item in q)
{
   Console.WriteLine(item);
}
```

演示示例：SkipWhile/TakeWhile 方法示例

分区类查询方法小结

请判断以下代码输出结果是什么？

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var q = numbers.Skip(1).Take(3).Skip(1).Take(2);
foreach (var item in q)
{
   Console.WriteLine(item);
}
```

输出：3 4

集合类查询方法

集合类查询方法

- Distinct

Distinct 示例

Distinct 去掉集合中的重复项

```
int[] factorsOf300 = { 2, 2, 3, 5, 5 };

var uniqueFactors = factorsOf300.Distinct();
```

输出: 2 3 5

演示示例： Distinct 方法示例

生成类查询方法

生成类查询方法

- Range
- Repeat

Range 示例

Range 生成一个整数序列

```
var numbers =Enumerable.Range(1, 10);
foreach (var item in numbers)
{
   Console.WriteLine(item);
}
```

演示示例：Range 方法示例

Repeat 示例

Repeat 生成一个重复项的序列

```
var numbers =
            Enumerable.Repeat(“Beijing 2008”, 10);
foreach (var item in numbers)
{
   Console.WriteLine(item);
}
```

演示示例：Repeat 方法示例

生成类查询方法小结

使用生成类查询方法时，需要注意以下几点:

- 和其他几类方法不同，Range/Repeat 不是扩展方法，而是普通的静态方法
- Range 只能产生整数序列
- Repeat 可以产生泛型序列
- 所有的查询方法都存放在 System.Linq.Enumerable 静态类中

高级查询方法小结

请判断以下代码输出结果是什么？

```
List<Person> foxRiver8 = GetFoxRiver8();

var q = from p in foxRiver8
        orderby p.FirstName thenby p.LastName
        select p;

foreach (var item in q)
{
   Console.WriteLine(item);
}
```

代码错误，查询语句中没有 thenby 关键字

总结

- 使用什么方法可以使查询在定义时就立即执行?
- 查询方法和查询语句是一一对应关系么?
- ThenBy 方法的主要作用是什么?
- 查询方法位于哪个名字空间下的哪个类中?