LINQ 基础

回顾

- 请说出三条以上 C# 3.0 的新特性
- 请说出扩展方法定义的语法格式
- 使用 var 和 object 声明变量有什么区别?

本章目标

- 掌握 LINQ 中的基本概念
- 掌握 LINQ 的组成
- 理解委托和匿名方法
- 理解 Lambda 表达式
- 掌握基本查询方法
  - Select()
  - Where()
  - OrderBy()
  - GroupBy()

LINQ要解决的问题

长期以来，开发社区形成以下格局：

- 面向对象与数据访问两个领域长期分裂，各自为政
- 编程语言中的数据类型与数据库中的数据类型形成两套体系。例如：
  - C# 中
  - SQL
- SQL 编码体验落后
  - 没有智能感应
  - 没有严格意义上的强类型和类型检查
- SQL 和 XML 都有各自的查询语言，而对象没有自己的查询语言

LINQ 将改变这一切！

LINQ 的历史

- 最初由 Anders Hejlsberg 构思，最初的研究计划称为 Cω
- 2005年9月 – 第一个为 C# 2.0 开发的技术预览版在当年的 PDC（微软开发者大会）上发布
- 2005年11月 – 更新至社区预览版（C# 2.0）
- 2006年1月 – 第一个为 VB 8.0 开发的技术预览版发布
- 2007年11月19日 – LINQ作为 .NET Framework 3.5 的一部分正式发布

Anders Hejlsberg

LINQ是什么

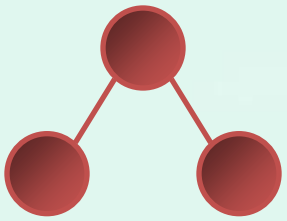
- LINQ（Language Integrated Query）即语言集成查询
- LINQ 是一组语言特性和API，使得你可以使用统一的方式编写各种查询。查询的对象包括XML、对象集合、SQL Server 数据库等等。
- LINQ 主要包含以下三部分：
  - LINQ to Objects 主要负责对象的查询
  - LINQ to XML 主要负责 XML 的查询
  - LINQ to ADO.NET 主要负责数据库的查询
    - LINQ to SQL
    - LINQ to DataSet
    - LINQ to Entities

LINQ 的组成

C# | VB | Others...

.NET Language Integrated Query

LNIQ to Objects | LINQ to ADO.NET | LINQ to XML



Objects

Relational

XML

LINQ 初体验

在没有LINQ以前，我们这样查询：

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

List<int> even = new List<int>();

foreach (int number in numbers)
{
   if (number % 2 == 0)
   {
      even.Add(number);
   }
}

even.Sort();
even.Reverse();
```

从 numbers 数组中提取偶数并降序排列

LINQ 初体验

今天，我们有了LINQ! 我们这样查询：

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

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

从 numbers 数组中提取偶数并降序排列

演示示例：Hello, LINQ

代码分析

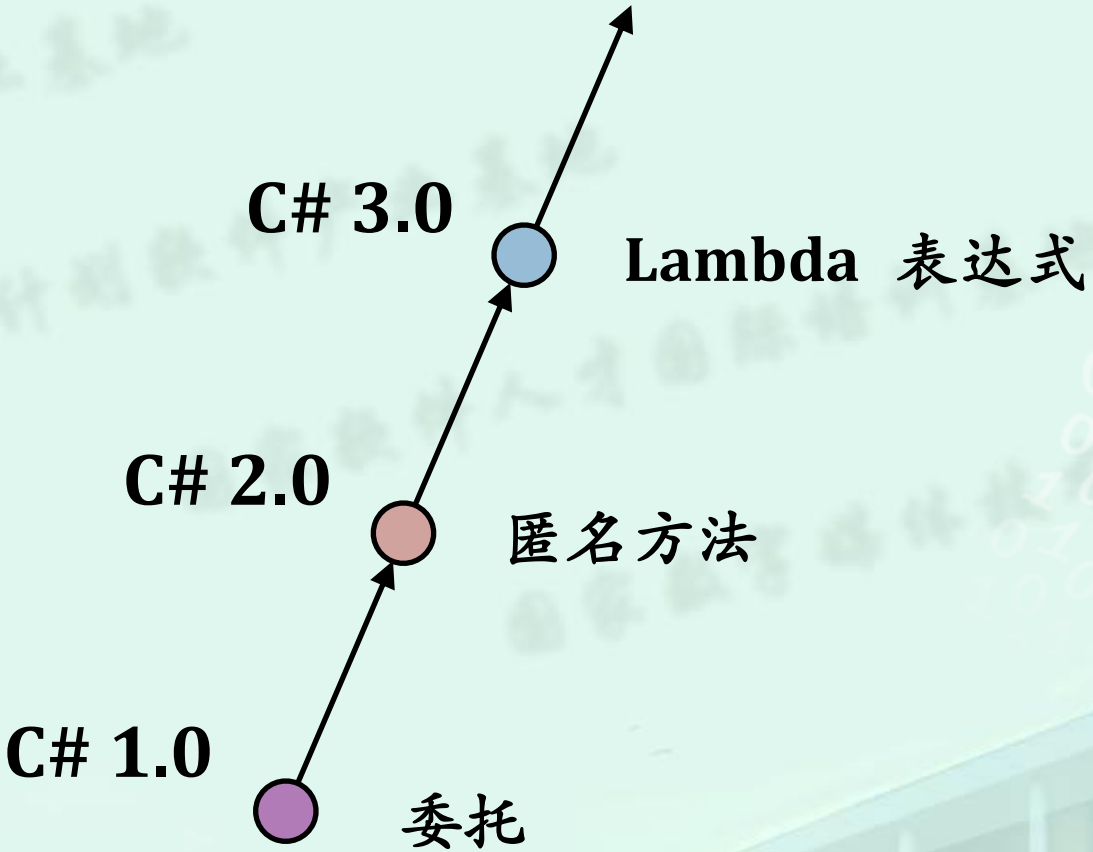

```
int[] numbers = new int[]
                { 6, 4, 3, 2, 9, 1, 7, 8, 5 };

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

推断类型

???

扩展方法

Lambda 表达式的进化



委托

委托的定义：

委托可以理解为一个函数指针，它定义了一个函数的原型

```
delegate string ProcessString( string input );
```

委托的实例化和使用：

委托的实例化就是将委托变量指向一个符合委托原型的实际方法

```
ProcessString p =
            new ProcessString( LowerIt );
foreach (string name in foxRiver8)
{
   Console.WriteLine( p ( name ) );
}
```

```
private string LowerIt
   (string input)
{
   return input.ToLower();
}
```

此时的 p 实际上就是 LowerIt() 方

演示示例：委托的使用

小结

- 委托可以看作是托管版本的函数指针
- 委托只对方法的原型（签名）进行约束
- 委托可以方便我们在程序运行时动态决定对象的行为

小结

- 委托和接口有什么区别?
- 我们在以前的课程中学习过哪些接口或委托?

匿名方法

在 C# 2.0 中，加入了匿名方法特性：

```
// 匿名方法方式
ProcessString p = delegate( string input )
{
   return input.ToLower();
};
foreach (string name in foxRiver8)
{
   Console.WriteLine( p ( name ) ) ;
}
```

参数列表

方法体

注意: 这里没有了具体的方法名称 因此称为匿名方法

演示示例： 匿名方法的使用

Lambda 表达式

在 C# 3.0 中，继匿名方法之后加入了更为简洁的 Lambda 表达式：

```
// Lambda 表达式方式
ProcessString p = input => input.ToLower();
foreach (string name in foxRiver8)
{
   Console.WriteLine( p ( name ) ) ;
}
```

演示示例：Lambda表达式的使用

Lambda表达式语法

最基本的 Lambda 表达式语法如下：

```
(参数列表) => {方法体}
```

说明

- 参数列表中的参数类型可以是明确类型或者是推断类型
- 如果是推断类型，则参数的数据类型将由编译器根据上下文自动推断出来

Lambda表达式的简写方式

如果参数列表只包含一个推断类型参数时

```
(参数列表) => {方法体}
```

```
参数列表 => {方法体}
```

进行以下转换的前提是此处 x 的数据类型可以根据上下文推断出来

```
(int x) => {return x+1;}
```

```
x => {return x+1;}
```

Lambda表达式的简写方式

如果方法体只包含一条语句时

(参数列表) => {方法体}

(参数列表) => 表达式

(int x) => {return x+1;}

(int x) => x+1

Lambda表达式的更多例子

多参数，推断类型参数列表，表达式方法体

```
(x, y) => x * y
```

无参数，表达式方法体

```
() => Console.WriteLine()
```

多参数，推断类型参数列表，多语句方法体

```
(x, y) => {
            Console.WriteLine( x );
            Console.WriteLine( y );
}
```

Lambda与匿名方法的关系

总体上说，匿名方法可以看作是Lambda 表达式的功能子集，但是两者存在以下区别：

- Lambda 表达式的参数允许不指明参数类型，而匿名方法的参数必须明确指明参数类型
- Lambda 表达式的方法体允许由单一表达式或者多条语句组成，而匿名方法不允许单一表达式形式

基本查询操作符-获取数据

Select()

```
public static IEnumerable<TResult> Select<TSource, TResult> (
        this IEnumerable<TSource> source,
        Func<TSource, TResult> selector )
```

说明

- Select 方法本身是一个泛型扩展方法
- 它作用于IEnumerable<TSource>类型
- 它只接受一个 Func<TSource, TResult> 类型参数
- Func<TSource, TResult> 是一个泛型委托，位于System名字空间下，System.Core.dll中
- 在这里 selector 是一个提取器

Select() 例子

以 Lambda 表达式形式出现的 Func<TSource, TResult>委托实例

```
var q1 = foxRiver8.Select
         (name => name.ToLower());

foreach (var item in q1)
{
   Console.WriteLine(item);
}
```

演示示例：Select方法示例

基本查询操作符-过滤数据

Where()

```
public static IEnumerable<TSource> Where<TSource>(
        this IEnumerable<TSource> source,
        Func<TSource, bool> predicate )
```

说明

- Where方法也是一个泛型扩展方法
- 它和 Select() 一样作用于IEnumerable<TSource>类型
- 它只接受一个 Func<TSource, bool> 泛型委托参数
- 在这里 predicate 是一个判断条件

Where() 例子


```
var q2 = foxRiver8
   .Where( name => name.StartsWith("T"))
   .Select(name => name.ToLower());

foreach (var item in q2)
{
   Console.WriteLine(item);
}
```

以 Lambda 表达式形式出现的判断条件，注意返回值要求为 bool 类型

演示示例：Where方法示例

基本查询操作符-排序数据

OrderBy()

```
public static IOrderedEnumerable<TSource> OrderBy<TSource,
TKey>(   this IEnumerable<TSource> source,
            Func<TSource, TKey> keySelector )
```

说明

- OrderBy方法也是一个泛型扩展方法
- 它和 Select() 一样作用于IEnumerable<TSource>类型
- 它只接受一个 Func<TSource, TKey > 类型参数
- 在这里 keySelector 指定要排序的字段
- 如果想降序排列可以使用OrderByDescending方法

OrderBy() 例子

排序字段，这里指定按照姓名的第二个字母升序排列

```
var q3 = foxRiver8
   .Where(name => name.Length >5)
   .Select(name => name.ToLower())
   .OrderBy(name => name.Substring(1,1))

foreach (var item in q3)
{
   Console.WriteLine(item);
}
```

演示示例：OrderBy方法示例

基本查询操作符-分组数据

请注意这个返回值与前面方法的返回值的区别

GroupBy()

```
public static IEnumerable<IGrouping<TKey, TSource>>
GroupBy<TSource, TKey>(
        this IEnumerable<TSource> source,
        Func<TSource, TKey> keySelector )
```

说明

- GroupBy方法和OrderBy方法非常类似，它也是一个泛型扩展方法
- 它和 OrderBy() 一样作用于IEnumerable<TSource>类型
- 它只接受一个 Func<TSource, TKey > 类型参数
- 在这里 keySelector 指定要分组的字段

GroupBy() 例子


```
var q4 = foxRiver8
   .Where(name => name.Length > 5)
   .Select(name => name.ToLow
   .GroupBy(name => name.Sub

foreach (var group in q4)
{
   Console.WriteLine(group.Key);
   Console.WriteLine("--------------------");
   foreach (var item in group)
   {
      Console.WriteLine(item);
   }
}
```

外层循环得到分组

内层循环得到分组中的项

演示示例：GroupBy方法示例

LINQ 的更多资源

- 官方网站
  - http://msdn2.microsoft.com/zh-cn/netframework/aa904594(en-us).aspx
- Hooked On LINQ
  - http://www.hookedonlinq.com/

总结

- LINQ 技术主要有哪几部分组成?
- 以下Lambda表达式表示什么意思?

```
(a, b) => a + b
```

- 本章所学的 Select Where OrderBy GroupBy 方法是作用于哪个类型的扩展方法?