

编写LINQ查询

北京理工大学计算机学院
金旭亮

基本技巧

集成编程语言特性查询数据

示例: 在C盘上查找昨天以前修改过的纯文本文件

```
IEnumerable<FileInfo> files =  
    from fileName in Directory.GetFiles("C:\\")  
    where File.GetLastWriteTime(fileName) < DateTime.Now.AddDays(-1)  
           && Path.GetExtension(fileName).ToUpper() == ".TXT"  
    select new FileInfo(fileName);
```

可以在where子句中直接使用C#的运算符“&&”（“||”和“!”也可以使用）。

可以在select子句中转换数据类型，动态创建新的对象。

混用LINQ查询与扩展方法

示例：编程列出Enumerable类型的所有公共成员

```
MemberInfo[] members = typeof(Enumerable).GetMembers(  
    BindingFlags.Static | BindingFlags.Public);  
  
var methods = (from method in members  
    select method.Name).Distinct();
```

要点：

可以使用**Distinct()**扩展方法消除查询结果集中的重复项，.NET基类库中针对IEnumerable<T>定义的其他扩展方法也可以使用。

在orderby子句中使用多字段排序

示例：查找C盘根目录下所有的文件，先按文件大小，再按文件名字排序。

```
IEnumerable<string> fileNames =  
    from fileName in Directory.GetFiles("C:\\")  
    orderby (new FileInfo(fileName)).Length,  
            fileName ascending  
    select fileName;
```

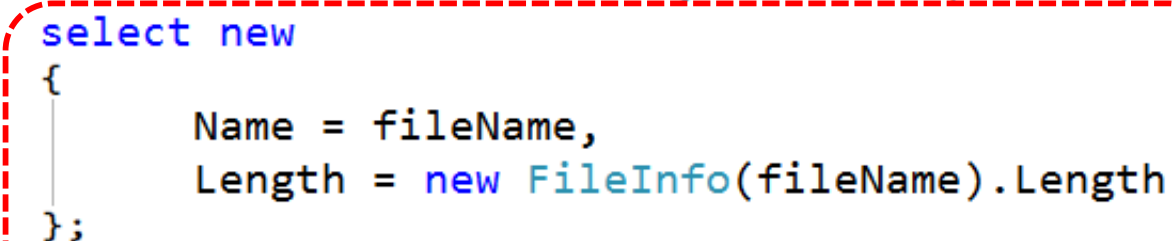
降序为descending

编译时，orderby子句被转换为对**OrderBy**扩展方法的调用，orderby子句中的多个键转换为对**ThenBy**扩展方法的调用。

使用匿名对象封装感兴趣的数据

示例：提取C盘根目录下的文件名与文件大小两个信息。

```
var files = from fileName in Directory.GetFiles("C:\\")
            select new
            {
                Name = fileName,
                Length = new FileInfo(fileName).Length
            };
```



注意：匿名对象创建之后，是只读的。

通过在编程时“动态”创建一个匿名对象，使我们无需创建类就可以封装感兴趣的数据，有助于避免实际开发中“类”数目的增长过快，这个技巧在开发中常用。

引入新的范围变量暂存查询结果

对于一些比较复杂的查询，可以引入一个新的范围变量来暂存子查询的结果，从而简化LINQ查询语句的编写。

```
IEnumerable<FileInfo> files =  
    from fileName in Directory.GetFiles("C:\\")  
    let file = new FileInfo(fileName)  
    orderby file.Length, fileName  
    select file;
```

上述语句引入新的范围变量`file`来表示根据`fileName`创建的`FileInfo`对象，并在后面的子句中直接使用这一中间变量，避免了在LINQ查询表达式中多次重复“`new FileInfo(fileName)`”。

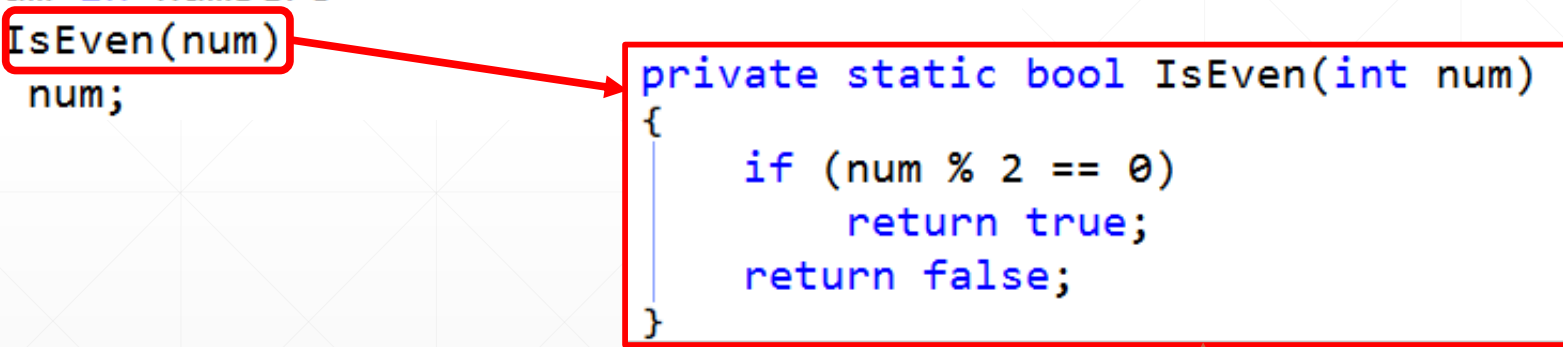
在LINQ中直接调用本地方法

// 数据源.

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
```

//查找所有偶数

```
var queryEvenNums =  
    from num in numbers  
    where IsEven(num)  
    select num;
```

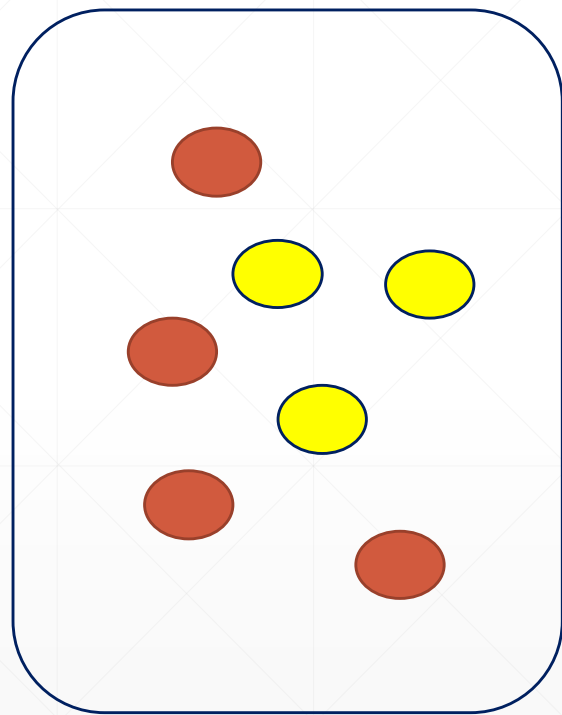


```
private static bool IsEven(int num)  
{  
    if (num % 2 == 0)  
        return true;  
    return false;  
}
```

C#编写的本地方法

分组

什么叫数据分组？



原始数据



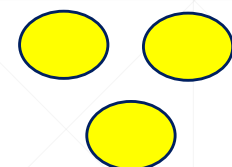
分组依据（颜色）

第1组

黄色

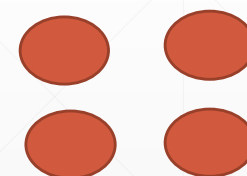


每组中的数据

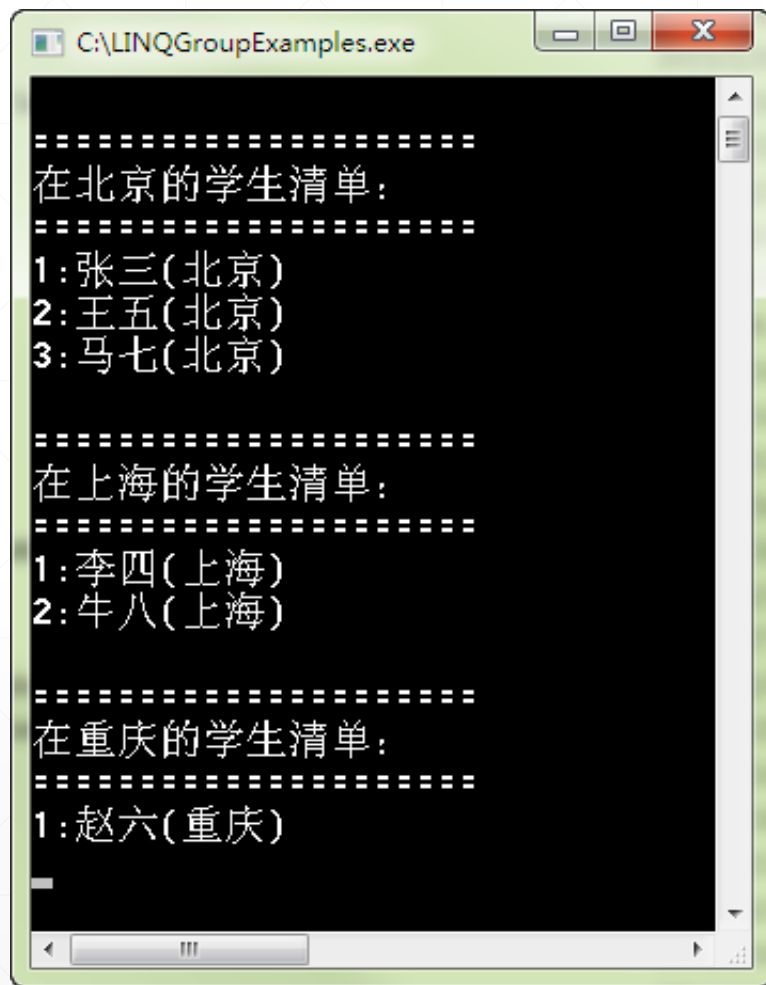


第2组

红色



“分组”引例



```
=====
在北京的学生清单:
=====
1: 张三(北京)
2: 王五(北京)
3: 马七(北京)

=====
在上海的学生清单:
=====
1: 李四(上海)
2: 牛八(上海)

=====
在重庆的学生清单:
=====
1: 赵六(重庆)
```

示例：LINQGroupExamples

LINQ 中实现分组的基本方式

```
// studnetQuery类型为 : IEnumerable<IGrouping<string, Student>>  
var studnetQuery = from student in students  
                   group student by student.City;
```

分组依据

分组中的单个数据对象

- .NET使用**IGrouping<TKey, TElement>**来表示一个分组。

分组数据的读取

要想显示分组后的所有数据，需要嵌套两层循环.....

```
//studentGroup类型为:IGrouping<string, Student>
```

```
foreach (var studentGroup in studnetQuery)
```

第一层循环，遍历
每个组

```
{  
    Console.WriteLine("\n=====");
```

```
    Console.WriteLine("在{0}的学生清单:", studentGroup.Key);
```

```
    Console.WriteLine("=====");
```

```
    int count = 0;
```

```
    foreach (Student stu in studentGroup)
```

第二层循环，遍历
每个组中的元素

```
{
```

```
    count++;
```

```
    Console.WriteLine("{0}:{1}({2})", count,  
        stu.Name, stu.City);
```

```
}
```

```
}
```

使用into子句针对分组进行进一步处理

// 按5个原音字母a、e、i、o、u将单词分组

```
var wordGroups =  
    from w in words  
    group w by w[0] into grps  
    where (grps.Key == 'a' || grps.Key == 'e'  
           || grps.Key == 'i' || grps.Key == 'o' || grps.Key == 'u')  
    select grps;
```

Into后面的“变量”用于引用分组结果

实现“非此即彼”的分组

```
C:\LINQGroupExamples.exe

成绩优秀的学生
张三, 平均分: 77.5
王五, 平均分: 93.5
赵六, 平均分: 75.5
马七, 平均分: 88.25

有不及格课程的学生
李四, 平均分: 72.25
```

To be, or not to be:
that is the question.

编写返回true或false的本地方法，
在LINQ表达式中直接调用并将其作为
分组依据。

```
List<Student> students = GetStudents();

// 按true或false分组
// 查询结果类型为: IEnumerable<IGrouping<bool, Student>>
var booleanGroupQuery =
    from student in students
    group student by HasFailed(student.Scores);

//输出查询结果
foreach (var studentGroup in booleanGroupQuery)
{
    Console.WriteLine();
    Console.WriteLine(studentGroup.Key ==
        true ? "有不及格课程的学生" : "成绩优秀的学生");
    foreach (var student in studentGroup)
    {
        Console.WriteLine("    {0}, 平均分: {1}",
            student.Name, student.Scores.Average());
    }
}
```

```
static bool HasFailed(List<int> Scores)
{
    foreach (int score in Scores)
    {
        if (score < 60)
            return true;
    }
    return false;
}
```

直接调用.NET基类库的Average扩展方法可以
计算出平均分

多段分组



```
C:\LINQGroupExamples.exe
中:
  张三, 平均分: 77.5
  李四, 平均分: 72.25
  赵六, 平均分: 75.5
优:
  王五, 平均分: 93.5
良:
  马七, 平均分: 88.25
```

按考试分数将学生成绩数据分为“优”、“良”、“中”、“及格”和“不及格”几大类

```

public static string GroupKey(List<int> Scores)
{
    //计算平均分，并取整
    int avg = (int)Scores.Average();

    string ret = "";
    switch (avg / 10)    //整除以10
    {
        case 10:
        case 9:
            ret = "优";
            break;
        case 8:
            ret = "良";
            break;
        case 7:
            ret = "中";
            break;
        case 6:
            ret = "及格";
            break;
        default:
            ret = "不及格";
            break;
    }
    return ret;
}

```

可以采用类似于“非此即彼”的方法使用本地方法实现。

只需编写一个返回分类标准的本地方法，让其成为分组依据即可。

```

// 查询结果类型为：IEnumerable<IGrouping<string, Student>
var booleanGroupQuery =
    from student in students
    group student by GroupKey(student.Scores);

```

分组编程练习

1

使用LINQ查询指定文件夹下的所有文件，按其类型进行分组，在每组中按文件大小进行排序，显示在屏幕上

2

编写一个程序，它可以读入一个英文段落，计算出每个单词的出现频率（即：单词的出现次数/文章总单词数），并对出现频率进行降序排列，输出结果。

3

编写一个Person类，其中有一个年龄属性。创建一个Person对象的集合，然后按照年龄分为五组：老年、中年、青年、少年和儿童，然后分组输出。