LINQ——查询概述

简介

- 什么是LINQ?

LINQ查询表达式是LINQ很重要的一部分内容。它可以从一个或多个给定的数据源中检索数据，并指定检索结果的数据类型和表现形式。

LINQ查询表达式由一个或多个LINQ查询子句按照一定的规则组成

LINQ查询子句概述

- 查询（Query）是一组指令，这些指令可以从一个或多个给定的数据源中检索数据，并指定检索结果的数据类型和表现形式。
- 查询表达式是一种用查询语法表示的表达式，由一组用类似于SQL的声明性语法编写的子句组成。
- 每一个子句可以包含一个或多个C#表达式，而这些表达式本身又可能是查询表达式或包含查询表达式。
- 查询表达式和其他表达式一样，可以用在C#表达式有效的任何上下文中

常用子句

- LINQ查询表达式包含8个常用子句，如from子句、where子句、select子句等：

子句	说明
from子句	指定查询操作的数据源和范围变量。
where子句	筛选元素的逻辑条件，一般由逻辑运算符（如逻辑“与”、逻辑“或”）组成。
select子句	指定查询结果的类型和表现形式。
orderby子句	对查询结果进行排序，可以为“升序”或“降序”。
group子句	对查询结果进行分组。
into子句	提供一个临时标识符。该标识可以充当对join、group或 select子句的结果的引用。
join子句	连接多个查询操作的数据源。
let子句	引入用于存储查询表达式中的子表达式结果的范围变量。

子句说明

- LINQ查询表达式必须以from子句开头，并且必须以select或group子句结束。
- 在第一个from子句和最后一个select或group子句之间，查询表达式可以包含一个或多个where、orderby、group、join、let子句，甚至from子句。
- 另外，join和group子句还可以使用into子句指定临时标识符号

  - 代码实例：

```
int[] values = {0,1,2,3,4,5,6,7,8,9};
var value = from v in values
where v < 3
select v;
```

基本子句

- from子句
- where子句
- select子句
- group子句
- orderby子句
- into子句
- join子句
- let子句

Form子句

- LINQ查询表达式必须包含from子句，且以from子句开头。如果该查询表达式还包含子查询，那么子查询表达式也必须以from子句开头。from子句指定查询操作的数据源和范围变量。其中，数据源不但包括查询本身的数据源，而且还包括子查询的数据源。范围变量一般用来表示源序列中的每一个元素。下面的代码实例就演示了一个简单的查询操作，该查询操作从values数组中查询小于3的元素。其中，v为范围变量，values是数据源。

```
int[] values = {0,1,2,3,4,5,6,7,8,9};
var value = from v in values
where v < 3
select v;
```

- 注意：from子句指定的数据源的类型必须为IEnumerable、IEnumerable<T>或前两者的派生类型

where子句

- 在LINQ查询表达式中，where子句指定筛选元素的逻辑条件，一般由逻辑运算符（如逻辑“与”、逻辑“或”）组成。一个查询表达式可以不包含where子句，也可以包含1个或多个where子句。每一个where子句可以包含1个或多个布尔条件表达式。
  - int[] values = {0,1,2,3,4,5,6,7,8,9};
  - var value = from v in values
  - where v < 3
  - select v;
- 下面的代码实例就演示了一个简单的查询操作，并在该查询表达式中使用了where子句。其中，where子句由两个布尔表达式和逻辑符合“&&”组成。

select子句

- 在LINQ查询表达式中，select子句指定查询结果的类型和表现形式。LINQ查询表达式要么以select子句结束，要么为group子句结束。
- 下面的代码实例就演示了包含最简单select子句的查询操作。该select子句选择元素的本身。
  - int[] values = {0,1,2,3,4,5,6,7,8,9};
  - var value = from v in values
  - where v < 3
  - select v;

- 下面的代码实例就演示了使用select子句创建一个string类型的序列。其中，序列的值为users数据源元素的Username属性的值。该select子句选择元素的Username属性的值。

```
///构建数据源
List<UserInfo> users = new List<UserInfo>();
for (int i = 1; i < 10; i++)
{
        users.Add(new UserInfo(i, "User0" + i.ToString(),"User0" + i.ToString() + "@web.com"));
}
///查询ID值小于3的用户
IEnumerable<string> = from u in users
                where u.ID < 3
                select u.Username;
```

- 下面的代码实例就演示了使用select子句创建一个序列。其中，序列包含ID和Username两个属性。该select子句使用new语句创建一个新类型的序列，该新类型包含ID和Username两个属性。
- ///构建数据源
  - List<UserInfo> users = new List<UserInfo>();
  - for (int i = 1; i < 10; i++)
  - {
  - users.Add(new UserInfo(i, "User0" + i.ToString(),"User0" + i.ToString() + "@web.com"));
  - }
  - ///查询ID值小于3的用户
  - var values = from u in users
  - where u.ID < 3
  - select new{u.ID,u.Username};

group子句

- 在查询表达式中，group子句对查询的结果进行分组，并返回元素类型为IGrouping<TKey,TElement>的对象序列。
- 注意：TKey指定IGrouping<TKey,TElement>的键的类型，TElement指定IGrouping<TKey,TElement>的值的类型。访问IGrouping<TKey,TElement>类型的值的方法与访问IEnumerable<T>的元素的方式非常相似，在此不做详细介绍。

- 下面的代码实例中的GroupQuery()函数演示了group子句对查询的结果进行分组的方法，具体步骤说明如下。
  - （1）创建数据类型为List<UserInfo>的数据源users。
  - （2）使用group子句对结果进行分组。其中，根据用户名称（Username属性的值）的序号的奇偶进行分组。
  - （3）使用嵌套foreach语句输出查询的结果。
- 注意：查询结果values的数据类型为IEnumerable<IGrouping<bool,UserInfo>>。因此，输出查询结果信息语句需要使用两个foreach语句。第一个foreach语句得到IGrouping<bool,UserInfo>类型的元素，第二个foreach语句得到UserInfo类型的元素。

orderby子句

- 在LINQ查询表达式中，orderby子句可以对查询结果进行排序。排序方式可以为“升序”或“降序”，且排序的键可以为一个或多个。
- 注意：LINQ查询表达式对查询结果的默认排序方式为“升序”。

- 下面的代码实例中的OrderQuery()函数演示了orderby子句对查询的结果进行倒序排序的方法，具体步骤说明如下。
  - （1）创建数据类型为List<UserInfo>的数据源users。
  - （2）使用where子句选择ID值小于6的用户。
  - （3）使用orderby子句对查询结果按照用户的名称进行倒序排序。
  - （4）使用foreach语句输出查询的结果。

into子句

- 在LINQ查询表达式中，into子句可以创建一个临时标识符，使用该标识符可以存储group、join或select子句的结果。下面的代码实例中的GroupOtherQuery()函数演示了group子句对查询的结果进行分组的方法，具体步骤说明如下。
  - （1）创建数据类型为List<UserInfo>的数据源users。
  - （2）使用group子句对结果进行分组。其中，根据用户名称（Username属性的值）的序号的奇偶进行分组。
  - （3）使用into子句创建临时标识符g存储查询结果。
  - （4）使用where子句筛选组包含元素的数量大于等于5的组。
  - （5）使用嵌套foreach语句输出查询的结果。

join子句

- 在LINQ查询表达式中，join子句比较复杂，它可以设置两个数据源之间的关系。当然，这两个数据源之间必须存在相关联的属性或值。join子句可以实现以下3种联接关系。
  - ● 内部联接，元素的联接关系必须同时满足被联接的两个数据源。
  - ● 分组联接，含有into子句的join子句。
  - ● 左外部联接。

- 1．内部联接
  - 内部联接要求元素的联接关系必须同时满足被联接的两个数据源，和SQL语句中的INNER JOIN子句相似。下面的代码实例中的InnerJoinQuery()函数演示了join子句内部联接users和roles数据源的查询方法，具体步骤说明如下。
  - （1）创建两个数据源：users和roles。其中，users数据源的数据类型为List<UserInfo>，roles数据源的数据类型为List<RoleInfo>。
  - （2）使用where子句筛选元素的ID值小于9的元素。
  - （3）使用join子句内部联接roles数据源，联接关系为“相等”。
  - （4）使用foreach语句输出查询的结果。

- 2．分组联接
  - 含有into子句的join子句被分组联接。分组联接产生分层数据结构，它将第一个集合中的每个元素与第二个集合中的一组相关元素进行匹配。在查询结果中，第一个集合中的元素都会出现在查询结果中。如果第一个集合中的元素在第二个集合中找到相关元素，则使用被找到的元素，否则使用空。
  - 下面的代码实例中的GroupJoinQuery()函数演示了join子句分组联接users和roles数据源的查询方法，具体步骤说明如下。
  - （1）创建两个数据源：users和roles。其中，users数据源的数据类型为List<UserInfo>，roles数据源的数据类型为List<RoleInfo>。
  - （2）使用where子句筛选元素的ID值小于9的元素。
  - （3）使用join子句分组联接roles数据源，联接关系为“相等”，组的标识符为“g”。
  - （4）使用select子句查询一个新类型的数据。其中，Roles属性的值为分组g的值，它包含第一个集合的元素（用户）相关角色的列表。
  - （5）使用foreach语句输出查询的结果。

- 3.左外部联接
  - 左外部联接与SQL语句中的LEFT JOIN子句比较相似，它将返回第一个集合中的每一个元素，而无论该元素在第二个集合中是否具有相关元素。
  - 注意：LINQ查询表达式若要执行左外部联接，往往与DefaultIfEmpty()方法与分组联接结合起来使用。如果第一个集合中的元素没有找到相关元素时，DefaultIfEmpty()方法可以指定该元素的相关元素的默认元素。
  - 下面的代码实例中的LeftOutJoinQuery()函数演示了join子句左外部联接users和roles数据源的查询方法，具体步骤说明如下。
  - （1）创建两个数据源：users和roles。其中，users数据源的数据类型为List<UserInfo>，roles数据源的数据类型为List<RoleInfo>。
  - （2）使用where子句筛选元素的ID值小于9的元素。
  - （3）使用join子句分组联接roles数据源，联接关系为“相等”，组的标识符为“gr”。
  - （4）使用from子句选择gr分组的默认元素。
  - （5）使用select子句查询一个新类型的数据。其中，Roles属性的值为分组gr的值，它包含第一个集合的元素（用户）相关角色的列表。
  - （6）使用foreach语句输出查询的结果。

let子句

- 在LINQ查询表达式中，let子句可以创建一个新的范围变量，并且使用该变量保存表达式中的结果。let子句指定的范围变量的值只能通过初始化操作进行赋值，范围变量的值一旦被初始化，将不能再被改变。
- 下面的代码实例中的LetQuery()函数演示了let子句查询的方法，具体步骤说明如下。
  - （1）创建数据类型为List<UserInfo>的数据源users。
  - （2）使用let子句创建number范围变量，并初始化为用户名称中的序列号。
  - （3）where子句查询ID值小于9、且用户名称中的序列号（存储在number范围变量中）为偶数的用户。
  - （4）使用foreach语句输出查询的结果。

总结

- from子句　　指定查询操作的数据源和范围变量。
- where子句　　筛选元素的逻辑条件，一般由逻辑运算符（如逻辑“与”、逻辑“或”）组成。
- select子句　指定查询结果的类型和表现形式。
- orderby子句 对查询结果进行排序，可以为“升序”或“降序”。
- group子句　　对查询结果进行分组。
- into子句　　提供一个临时标识符。该标识可以充当对join、group或select子句的结果的引用。
- join子句　　连接多个查询操作的数据源。
- let子句　　引入用于存储查询表达式中的子表达式结果的范围变量。