# 其实就像数据库查询的一对多查询

# 执行分组联接（C# 编程指南）

分组联接可用于产生分层数据结构。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配对。

例如，一个名为 Student 的类或关系数据库表可能包含两个字段：Id 和 Name。另一个名为 Course 的类或关系数据库表可能包含两个字段：StudentId 和 CourseTitle。这两个数据源的分组联接（基于匹配的 Student.Id 和 Course.StudentId）会将每个 Student 与一个 Course 对象集合（可能为空）组合在一起。

| **https://wizardforcel.gitbooks.io/msdn-csharp/content/img/note.jpg 注意** |
| --- |
| 第一个集合中的每个元素都会出现在分组联接的结果集内，而无论是否在第二个集合中找到相关元素。如果未找到相关元素，则该元素的相关元素序列为空。因此，结果选择器可以访问第一个集合的每个元素。非分组联接中的结果选择器与此不同，它无法访问第一个集合中的那些在第二个集合中没有匹配元素的元素。 |

本主题中的第一个示例演示如何执行分组联接；第二个示例演示如何使用分组联接创建 XML 元素。

### 分组联接示例

下面的示例根据与 Pet.Owner 属性匹配的 Person 来对 Person 和 Pet 类型的对象执行分组联接。与为每个匹配产生一对元素的非分组联接不同，分组联接只为第一个集合的每个元素产生一个结果对象（在此示例中为一个 Person 对象）。第二个集合中的相应元素（在此示例中为 Pet 对象）被分组到一个集合中。最后，结果选择器函数为每个包含 Person.FirstName 和一个 Pet 对象集合的匹配创建一个匿名类型。

public class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

public class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

/// <summary>

/// This example performs a grouped join.

/// </summary>

public static void GroupJoinExample()

{

Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };

Person terry = new Person { FirstName = "Terry", LastName = "Adams" };

Person charlotte = new Person { FirstName = "Charlotte", LastName = "Weiss" };

Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };

Pet barley = new Pet { Name = "Barley", Owner = terry };

Pet boots = new Pet { Name = "Boots", Owner = terry };

Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };

Pet bluemoon = new Pet { Name = "Blue Moon", Owner = terry };

Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

// Create two lists.

List<Person> people = new List<Person> { magnus, terry, charlotte, arlene };

List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon, daisy };

// Create a list where each element is an anonymous type

// that contains the person's first name and a collection of

// pets that are owned by them.

var query = from person in people

join pet in pets on person equals pet.Owner into gj

select new { OwnerName = person.FirstName, Pets = gj };

foreach (var v in query)

{

// Output the owner's name.

Console.WriteLine("{0}:", v.OwnerName);

// Output each of the owner's pet's names.

foreach (Pet pet in v.Pets)

Console.WriteLine(" {0}", pet.Name);

}

}

// This code produces the following output:

//

// Magnus:

// Daisy

// Terry:

// Barley

// Boots

// Blue Moon

// Charlotte:

// Whiskers

// Arlene:

### 执行分组联接以创建 XML 的示例

分组联接非常适合于使用 LINQ to XML 来创建 XML。下面的示例与上一个示例类似，不同之处在于：结果选择器函数创建表示已联接对象的 XML 元素，而不是创建匿名类型。有关 LINQ to XML的更多信息，请参见[LINQ to XML](https://msdn.microsoft.com/zh-cn/library/bb387098.aspx)。

class Person

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

/// <summary>

/// This example creates XML output from a grouped join.

/// </summary>

public static void GroupJoinXMLExample()

{

Person magnus = new Person { FirstName = "Magnus", LastName = "Hedlund" };

Person terry = new Person { FirstName = "Terry", LastName = "Adams" };

Person charlotte = new Person { FirstName = "Charlotte", LastName = "Weiss" };

Person arlene = new Person { FirstName = "Arlene", LastName = "Huff" };

Pet barley = new Pet { Name = "Barley", Owner = terry };

Pet boots = new Pet { Name = "Boots", Owner = terry };

Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };

Pet bluemoon = new Pet { Name = "Blue Moon", Owner = terry };

Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

// Create two lists.

List<Person> people = new List<Person> { magnus, terry, charlotte, arlene };

List<Pet> pets = new List<Pet> { barley, boots, whiskers, bluemoon, daisy };

// Create XML to display the hierarchical organization of people and their pets.

XElement ownersAndPets = new XElement("PetOwners",

from person in people

join pet in pets on person equals pet.Owner into gj

select new XElement("Person",

new XAttribute("FirstName", person.FirstName),

new XAttribute("LastName", person.LastName),

from subpet in gj

select new XElement("Pet", subpet.Name)));

Console.WriteLine(ownersAndPets);

}

// This code produces the following output:

//

// <PetOwners>

// <Person FirstName="Magnus" LastName="Hedlund">

// <Pet>Daisy</Pet>

// </Person>

// <Person FirstName="Terry" LastName="Adams">

// <Pet>Barley</Pet>

// <Pet>Boots</Pet>

// <Pet>Blue Moon</Pet>

// </Person>

// <Person FirstName="Charlotte" LastName="Weiss">

// <Pet>Whiskers</Pet>

// </Person>

// <Person FirstName="Arlene" LastName="Huff" />

// </PetOwners>