

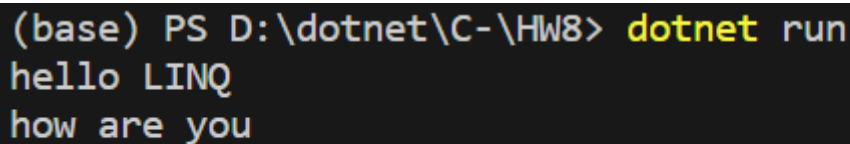
HW8 2150276 沈卓成

1 查询与语言结合

Code:

```
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] greetings = { "Hello", "hello LINQ", "how are you" };
            var items = from s in greetings
                        where s.Length > 5
                        select s;
            foreach (var item in items)
            {
                Console.WriteLine(item);
            }
            Console.ReadLine();
        }
    }
}
```

Ans:



```
(base) PS D:\dotnet\C-\HW8> dotnet run
hello LINQ
how are you

```

1. **定义字符串数组**: 首先, 创建了一个名为 `greetings` 的字符串数组, 其中包含三个字符串: "Hello", "hello LINQ", "how are you".
2. **LINQ查询**: 接着使用LINQ (Language Integrated Query, 语言集成查询) 来查询 `greetings` 数组。 `from s in greetings` 表示从 `greetings` 数组中选择每个元素 (此处用 `s` 表示)。 `where s.Length > 5` 是一个条件, 表示仅选择那些长度大于5的字符串。 `select s` 表示选择满足条件的字符串。整个查询的结果被赋值给 `items` 变量。
3. **遍历和打印**: 然后通过一个 `foreach` 循环遍历 `items` 中的每个项 (即每个长度大于5的字符串), 并使用 `Console.WriteLine(item);` 将它们打印到控制台上。
4. **等待输入**: 最后, `Console.ReadLine();` 使程序暂停运行, 等待用户输入。这通常用来防止控制台应用程序执行完毕后立即关闭窗口, 从而可以看到程序的输出结果。

2 创建可枚举对象

Code:

```
namespace ConsoleApp1
{
    public class MyEnumerator : IEnumerator
    {
        private int cur;
        private readonly MyEnumerable _owner;
        public bool MoveNext()
        {
            if (cur <= 0)
            {
                return false;
            }
            cur--;
            return true;
        }

        public void Reset()
        {
            cur = _owner.StartCountDown;
        }

        public object Current=>cur;

        public MyEnumerator(MyEnumerable countdown)
        {
            _owner = countdown;
            Reset();
        }
    }

    public class MyEnumerable : IEnumerable
    {
        public int StartCountDown;
        public IEnumerator GetEnumerator()
        {
            return new MyEnumerator(this);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            MyEnumerable countdownobj = new MyEnumerable{StartCountDown = 20};
            IEnumerator enu = countdownobj.GetEnumerator();
            while (enu.MoveNext())
            {
                int n =(int)enu.Current;
                Console.WriteLine(n);
            }

            foreach (var i in countdownobj)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

```
}  
}
```

Ans:

```
19  
18  
17  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1  
0  
19  
18  
17  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
5
```

1. `MyEnumerator` 类是一个迭代器，它实现了 `IEnumerator` 接口。这个类包含了如下的成员：
 - `cur`：当前计数器的值。
 - `_owner`：持有这个迭代器的 `MyEnumerable` 对象的引用。
 - `MoveNext` 方法：用于将迭代器推进到集合的下一个元素。如果迭代器已经到了集合的末尾（在这里即 `cur` 小于等于0），则返回 `false`。否则减少 `cur` 的值并返回 `true`。
 - `Reset` 方法：将迭代器重置到它的初始位置。在这个例子中，即将 `cur` 设置为 `_owner.StartCountDown` 的值。
 - `Current` 属性：获取集合中的当前元素，即当前 `cur` 的值。
2. `MyEnumerable` 类实现了 `IEnumerable` 接口。这个类只有一个属性 `StartCountDown`，表示倒计时开始的值，和一个方法 `GetEnumerator`，返回一个 `MyEnumerator` 对象。
3. `Program` 类包含程序的主入口点，即 `Main` 方法。在 `Main` 方法中，首先创建了一个 `MyEnumerable` 对象，设置其 `StartCountDown` 为20。然后通过 `GetEnumerator` 获取到对应的 `MyEnumerator` 对象，使用 `while` 循环将倒计时的值打印出来。最后，还演示了使用 `foreach` 循环遍历 `MyEnumerable` 对象的方式。通过实现 `IEnumerable` 和 `IEnumerator` 接口，`MyEnumerable` 对象能够被 `foreach` 循环正确遍历。

3 yield

Code:

```
using System.Collections;  
  
namespace ConsoleApp1  
{  
    public class Myyield : IEnumerable  
    {
```

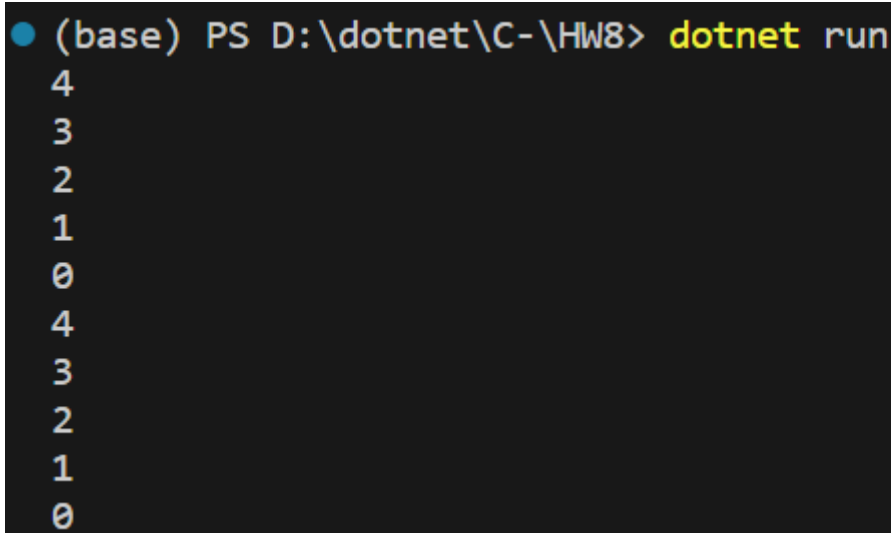
```

public int StartCount;
public IEnumerator GetEnumerator()
{
    for (int n = StartCount - 1; n >= 0; --n)
    {
        yield return n;
    }
}
}
class Program
{
    static void Main(string[] args)
    {
        MyYield myYield = new MyYield{StartCount = 5};
        IEnumerator enumerator = myYield.GetEnumerator();
        while (enumerator.MoveNext())
        {
            Console.WriteLine(enumerator.Current);
        }

        foreach (var i in myYield)
        {
            Console.WriteLine(i);
        }
    }
}

```

Ans:



```

(base) PS D:\dotnet\C-\HW8> dotnet run
4
3
2
1
0
4
3
2
1
0
4
3
2
1
0

```

1. `MyYield` 类实现了 `IEnumerable` 接口，这意味着该类的对象可以在 `foreach` 循环中被迭代。这个类有一个属性 `StartCount`，定义了倒数的起始值，和一个 `GetEnumerator` 方法，这个方法返回一个迭代器。

`GetEnumerator` 方法中的 `for` 循环实现了从 `StartCount-1` 倒数到 0。每次循环，它都使用 `yield return` 语句返回当前的 `n` 值。`yield` 关键字表示该方法是一个迭代器，当在 `foreach` 循环中使用该方法时，每次调用 `MoveNext` 方法时，C#运行时都会执行到下一个 `yield return` 语句，返回当前的值，并保存当前的执行状态，等待下一次调用 `MoveNext` 方法。

2. `Program` 类是程序的主入口点。在 `Main` 方法中，首先创建了一个 `MyYield` 对象，并设置其 `StartCount` 值为 5。然后使用 `GetEnumerator` 方法获取迭代器，并使用 `while` 循环将倒数的值打印出来。最后，它还演示了如何使用 `foreach` 循环直接迭代 `MyYield` 对象的方法。

4 LINQ

Code:

```
using System.Collections;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] strArray = { "one", "two", "three", "four", "five", "six",
            "seven", "eight", "nine", "ten" };
            // (1) 返回一个序列对象
            IEnumerable<string> items = strArray.Where(p => p.StartsWith('t'));

            foreach (var item in items)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("1. =====");
            // (2) 延迟执行
            strArray[2] = "t5000";
            foreach (var item in items)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("2. =====");
            // (3) 非延迟执行
            IEnumerable<string> item2 = strArray.Where(p =>
p.StartsWith('t')).ToList();
            foreach (var item in item2)
            {
                Console.WriteLine(item);
            }
            strArray[2] = "t6000";
            foreach (var item in item2)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("3. =====");
            // (4) 查看定义
            // public static IEnumerable<TSource> Where<TSource>(this
IEnumerable<TSource> source, Func<TSource, bool> predicate)
            // public static IEnumerable<TSource> Where<TSource>(this
IEnumerable<TSource> source, Func<TSource, int, bool> predicate) //
(5) 使用where的第二个原型
            IEnumerable<string> item3 = strArray.Where((p, i) =>
            {
                if (i > 2)
                {
                    return p.StartsWith('t');
                }
                else
                {
                    return false;
                }
            });
        }
    }
}
```

```

    }
});
foreach (var item in item3)
{
    Console.WriteLine(item);
}
Console.WriteLine("5. =====");
// (6) select
IEnumerable<int> item4 = strArray.Select(p => p.Length);
foreach (var item in item4)
{
    Console.WriteLine(item);
}
Console.WriteLine("6. =====");
// (7) select
var item5 = strArray.Select((p,i)=>new {Index = i,Name= p});
foreach (var item in item5)
{
    Console.WriteLine(item);
}
Console.WriteLine("7. =====");
// (8) selectMany
var items6 = strArray.SelectMany(p => p.ToCharArray());
foreach (var item in items6)
{
    Console.WriteLine(item);
}
Console.WriteLine("8. =====");
// (9) Take
var items7 = strArray.Take(3);
foreach (var item in items7)
{
    Console.WriteLine(item);
}
Console.WriteLine("9. =====");
// (10) TakeWhile
var items8 = strArray.TakeWhile(p => p.Length <= 4);
foreach (var item in items8)
{
    Console.WriteLine(item);
}
Console.WriteLine("10. =====");
// (11) skip
var items9 = strArray.Skip(3);
foreach (var item in items9)
{
    Console.WriteLine(item);
}
Console.WriteLine("11. =====");
// (12) SkipWhile
var items10 = strArray.SkipWhile(p => p.Length < 4);
foreach (var item in items10)
{
    Console.WriteLine(item);
}
Console.WriteLine("12. =====");
// (13) order by
var items11 = strArray.OrderBy(p => p.Length);

```

```
        foreach (var item in items11)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("13. =====");
        var items12 = strArray.OrderBy(p => p.Length).Take(5);
        foreach (var item in items11)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("14. =====");
    }
}
```

Ans(输出太长了，只截取一部分):

```

(base) PS D:\dotnet\C-\HW8> dotnet run
● two
three
ten
1. =====
two
t5000
ten
2. =====
two
t5000
ten
two
t5000
ten
3. =====
ten
5. =====
3
3
5
4
4
3
5
5
4
3
6. =====
{ Index = 0, Name = one }
{ Index = 1, Name = two }
{ Index = 2, Name = t6000 }
{ Index = 3, Name = four }
{ Index = 4, Name = five }
{ Index = 5, Name = six }
{ Index = 6, Name = seven }

```

1. 定义了一个字符串数组 `strArray` 包含了10个字符串。
2. (1) 返回一个序列对象：使用LINQ的 `where` 方法筛选出 `strArray` 中所有以字符't'开头的元素，并用 `foreach` 循环打印出这些元素。
3. (2) 延迟执行：LINQ的执行模式是延迟执行，也就是说查询定义后并不会立即执行，而是在迭代结果的时候执行。所以在修改了数组中的第三个元素后，再次迭代输出的结果也会受到修改的影响。
4. (3) 非延迟执行：通过 `ToList` 方法，查询结果会立即执行并保存到列表中，这时修改原数组并不会影响已保存的查询结果。
5. (4) 查看定义 (5) 使用`where`的第二个原型：`where` 方法有两个重载，第二个重载接受的委托同时包含元素和它的索引。在这个例子中，只有索引大于2且以't'开头的元素才会被选中。

6. (6) select: 使用 select 操作可以从元素中投射出需要的部分, 这里投射出了每个字符串的长度。
7. (7) select: select 也可以用来创建匿名类型, 这里创建了包含索引和名称的对象。
8. (8) selectMany: selectMany 用来展平集合, 这里将每个字符串的字符展平成了一个字符序列。

总结:

1. Take: 取序列的前n个元素。
2. TakeWhile: 取序列的前n个满足条件的元素。
3. Skip: 跳过序列的前n个元素。
4. SkipWhile: 跳过序列前面满足条件的元素。
5. OrderBy: 按照某个键对序列元素进行排序。

5 连接

Code:

```
using System.Collections;

namespace ConsoleApp1
{
    class Program
    {
        public class Student
        {
            public int Id;
            public string Name;
            public int Age;
        }

        public class StudentScore
        {
            public int StudentId;
            public float English;
            public float Math;
        }

        static void Main(string[] args)
        {
            Student[] students = new Student[]
            {
                new Student { Id = 1, Name = "John", Age = 18 },
                new Student { Id = 2, Name = "Marry", Age = 19 },
                new Student { Id = 3, Name = "Tom", Age = 20 },
                new Student { Id = 4, Name = "Jerry", Age = 21 },
                new Student { Id = 5, Name = "Jack", Age = 22 }
            };

            StudentScore[] score = new StudentScore[]
            {
                new StudentScore { StudentId = 1, English = 80, Math = 90 },
                new StudentScore { StudentId = 2, English = 85, Math = 95 },
                new StudentScore { StudentId = 3, English = 90, Math = 100 },
                new StudentScore { StudentId = 4, English = 95, Math = 105 },
                new StudentScore { StudentId = 5, English = 100, Math = 110 }
            };

            var list = students.Join(score, o => o.Id, i => i.StudentId,
```

```

(o,i)=> new {Name=o.Name, English = i.English, Math =i.Math});

        foreach (var item in list)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("=====");
        var list2 = from s in students
                    from sc in score
                    where s.Id == sc.StudentId
                    orderby sc.Math
                    select new{Name = s.Name,English = sc.English, Math
= sc.Math};
        foreach (var item in list2)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine("=====");
    }
}
}

```

Ans:

```

(base) PS D:\dotnet\C-\HW8> dotnet run
● D:\dotnet\C-\HW8\5.cs(10,27): warning CS8618: 在退出构造函数时,
  声明为可以为 null。 [D:\dotnet\C-\H
  W8\HW8.csproj]
{ Name = John, English = 80, Math = 90 }
{ Name = Marry, English = 85, Math = 95 }
{ Name = Tom, English = 90, Math = 100 }
{ Name = Jerry, English = 95, Math = 105 }
{ Name = Jack, English = 100, Math = 110 }
=====
{ Name = John, English = 80, Math = 90 }
{ Name = Marry, English = 85, Math = 95 }
{ Name = Tom, English = 90, Math = 100 }
{ Name = Jerry, English = 95, Math = 105 }
{ Name = Jack, English = 100, Math = 110 }
=====

```

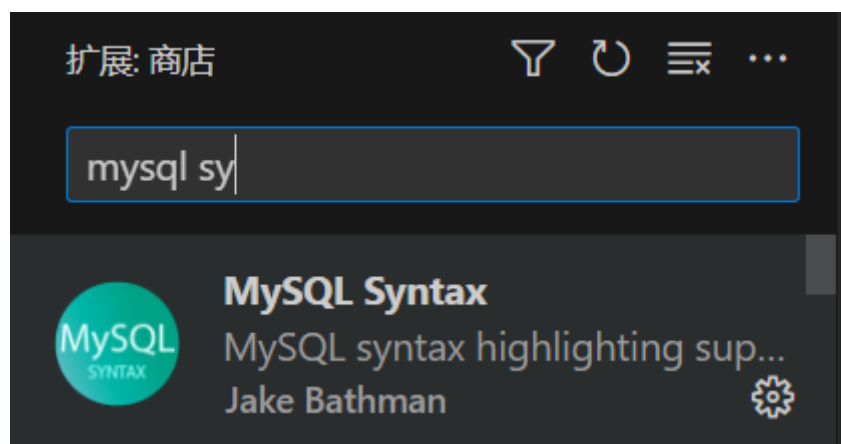
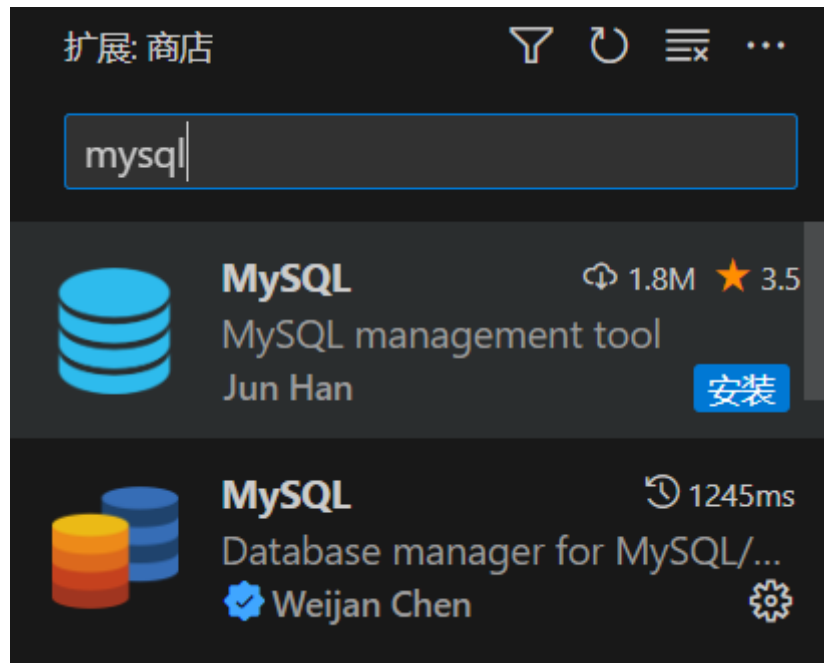
1. **Student 类**: 包含三个公共属性, 分别是Id (学生ID, 整数类型), Name (学生姓名, 字符串类型), 和Age (学生年龄, 整数类型)。
2. **StudentScore 类**: 包含三个公共属性, 分别是StudentId (学生ID, 整数类型, 用于与Student类的Id属性关联), English (英语成绩, 浮点数类型), 和Math (数学成绩, 浮点数类型)。
3. 在 Main 方法中:
 - 首先初始化了一个 Student 数组和一个 StudentScore 数组, 包含了几学生的基本信息和他们的成绩信息。
 - 使用LINQ的 Join 方法将 students 数组和 score 数组连接起来, 连接条件是两个数组中的 Id 和 StudentId 相等。查询结果是一个匿名类型的集合, 包含学生的姓名、英语成绩和数学成绩。
 - 使用 foreach 循环遍历上述连接查询的结果并打印到控制台。
 - 使用LINQ的查询表达式语法进行另一个查询, 这次是从 students 数组和 score 数组中获取每个学生的姓名、英语成绩和数学成绩。这个查询同样基于学生ID和学生成绩ID的匹配, 但

是额外增加了一个按数学成绩升序排序的条件。

- 再次使用 `foreach` 循环遍历查询结果并打印到控制台。

6 ADO.NET的使用

首先在vscode中下载mysql和mysql syntax插件



建表：

```
-- 创建学生表
CREATE TABLE students (
    student_id INT AUTO_INCREMENT PRIMARY KEY, -- 学生ID, 主键
    name VARCHAR(50) NOT NULL, -- 学生姓名
    age INT, -- 年龄
    gender ENUM('male', 'female') -- 性别
);

-- 创建学生成绩表
CREATE TABLE student_scores (
    id INT AUTO_INCREMENT PRIMARY KEY, -- 成绩记录ID, 主键
    student_id INT, -- 学生ID, 外键
    subject VARCHAR(50), -- 科目
    score DECIMAL(5, 2), -- 分数, 保留两位小数
    FOREIGN KEY (student_id) REFERENCES students(student_id) -- 外键约束, 连接到学生
    表的学生ID
);
```

模拟一些数据插入：

```
-- 向students表插入数据
INSERT INTO students (name, age, gender) VALUES ('张三', 20, 'male');
INSERT INTO students (name, age, gender) VALUES ('李四', 19, 'female');
INSERT INTO students (name, age, gender) VALUES ('王五', 21, 'male');

-- 向student_scores表插入数据
-- 假设张三、李四和王五的student_id分别是1、2、3
INSERT INTO student_scores (student_id, subject, score) VALUES (1, '数学', 90.5);
INSERT INTO student_scores (student_id, subject, score) VALUES (1, '英语', 88.0);
INSERT INTO student_scores (student_id, subject, score) VALUES (2, '数学', 85.0);
INSERT INTO student_scores (student_id, subject, score) VALUES (2, '英语', 92.5);
INSERT INTO student_scores (student_id, subject, score) VALUES (3, '数学', 78.0);
INSERT INTO student_scores (student_id, subject, score) VALUES (3, '英语', 81.0);
```

查询：

```
using System;
using System.Data;
using System.Data.SqlClient;

class Program
{
    static void Main()
    {
        // 这个连接字符串应该根据你的实际数据库配置来设置
        string connectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["MySQLConnection"].C
onnectionString;

        // SQL 查询语句，联合两个表
        string queryString =
            "SELECT s.name, sc.subject, sc.score " +
            "FROM students s " +
            "JOIN student_scores sc ON s.student_id = sc.student_id;";

        // 创建连接
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            // 创建SQL命令
            SqlCommand command = new SqlCommand(queryString, connection);
            connection.Open();

            // 执行命令并处理结果
            using (SqlDataReader reader = command.ExecuteReader())
            {
                Console.WriteLine("Name\tSubject\tScore");

                while (reader.Read())
                {
                    Console.WriteLine($"
{reader["name"]}\t{reader["subject"]}\t{reader["score"]}");
                }
            }
        }
    }
}
```

```
    }  
  
    Console.ReadLine(); // Pausing if running from console  
}  
}
```

结果：

该例子的输出是数据库中每个学生的姓名、科目和成绩：

Name	Subject	Score
张三	数学	90.5
张三	英语	88.0
李四	数学	85.0
李四	英语	92.5
王五	数学	78.0
王五	英语	81.0