C#语法专题（二）

厦门大学信息学院 赵江声

2023-09

目录

Content

01

可为空的值类型

基础值类型 T 本身不能是可为空的值类型。

1.1 基本概念

参考资料：https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/builtin-types/nullable-value-types

- 可为 null 值类型 T? 表示其基础值类型T 的所有值及额外的 null 值。
- 基础值类型 T 本身不可为空。

```
1  //int i = null;  //基本值类型不可为空，编译不能通过
2
3  //可为空值类型声明方式
4  double? pi = 3.14;
5  char? letter = 'a';
6
7  int n1 = 10;
8  int? n2 = n1;
9
10 bool? flag = null;
11
12 // An array of a nullable value type:
13 int?[] arr = new int?[10];
```


1.2 判断值为空的几种方法

- is 运算符与类型模式
- Nullable<T>.HasValue
- 与 null 进行比较

```
1  //判断值为空的三种方法
2  int? i = 1108;
3
4  //is 运算符与类型模式
5  string s = (i is int iValue) ? $"{i}:非空" : $"null:空";
6
7  //Nullable<T>.HasValue
8  s = (i.HasValue) ? $"{i}:非空" : $"null:空";
9
10 //与 null 进行比较
11 s = (i ≠ null) ? $"{i}:非空" : $"null:空";
```


1.3 从可为空的值类型转换为基础类型

- 如果要将可为空值类型的值分配给不可以为 null 的值类型变量，则可能需要指定要分配的替代 null 的值。使用 Null 合并操作符?? 执行此操作。

```
1 int? a = 28;
2 int b = a ?? -1;
3 Console.WriteLine($"b is {b}"); // output: b is 28
4
5 int? c = null;
6 int d = c ?? -1;
7 Console.WriteLine($"d is {d}"); // output: d is -1
```


02

Null 合并操作符

?? ??=

2.1 基本概念

- 参考资料：https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/operators/null-coalescing-operator
- Null 合并运算符 ??
  - 如果左操作数的值为 null，它会计算右操作数并返回其结果
  - 如果左操作数的值为 非null，则返回左操作数值
- Null 合并赋值运算符 ??=
  - 仅当左操作数为 null 时， 才会将其右操作数的值赋值给其左操作数。
  - 如果左操作数为非 null，则 ??= 运算符不会计算其右操作数。

2.2 样例

```
int? a = null;
//左操作数的值为 null，则计算右操作数并返回其结果
Console.WriteLine($"a??8 = {a ?? 8}");     //运行结果：8

a = 1;
//左操作数的值不为 null，则返回左操作数值
Console.WriteLine($"a??8 = {a ?? 8}");     //运行结果：1
```

```
int? b = null;
//仅当左操作数为 null 时， 才会将其右操作数的值赋值给其左操作数。
Console.WriteLine($"b??=8 = {b ??= 8}");      //此时b=8

//如果左操作数为非 null，则 ??= 运算符不会计算其右操作数。
Console.WriteLine($"b??=18 = {b ??= 18}");      //此时b=8
```


2.3 null 合并运算符是右结合运算符

```
1 a ?? b ?? c     //等同于 a ?? (b ?? c)
2
3 d ??= e ??= f    //等同于 d ??=  (e ??= f)
```


03

yield语句

提供下一个元素

3.1 基本概念

- 参考资料：https://learn.microsoft.com/zh-cn/dotnet/csharp/language-reference/statements/yield
- 在迭代器中使用yield 语句提供下一个值或表示迭代结束。有两种形式：yield return 和 yield break。
  - yield return ：在迭代中提供下一个值
  - yield break ：显式示迭代结束
- 迭代器的返回类型为 IEnumerable<T>（在非泛型情况下，使用 IEnumerable 作为迭代器的返回类型）。还可以使用 IAsyncEnumerable<T> 作为迭代器的返回类型。 这使得迭代器异步。 使用 await foreach 语句对迭代器的结果进行迭代

3.2 样例

```
IEnumerable<int> GetPositive(IEnumerable<int> numbers)
{
    foreach (int n in numbers)
    {
        if (n > 0)
        {
          yield return n;
        }
        else
        {
          yield break;
        }
    }
}

Console.WriteLine(string.Join(", ", GetPositive(new[] { 2, 8, 6, -1, 9, 0
                                                        })));
// Output: 2, 8, 6
```


3.3 迭代器异步

- 使用 IAsyncEnumerable<T> 作为迭代器的返回类型，这使得迭代器异步。 使用 await foreach 语句对迭代器的结果进行迭代

```
await foreach (int n in GenerateNumbersAsync(5))
{
    Console.Write(n);
    Console.Write(" ");
}
// Output: 0 2 4 6 8

async IAsyncEnumerable<int> GenerateNumbersAsync(int count)
{
    for (int i = 0; i < count; i++)
    {
        yield return await ProduceNumberAsync(i);
    }
}

async Task<int> ProduceNumberAsync(int seed)
{
    await Task.Delay(1000);
    return 2 * seed;
}
```


04

迭代器

遍历容器的对象

4.1 基本概念

- 参考资料：https://learn.microsoft.com/zh-cn/dotnet/csharp/iterators
- 迭代器是遍历容器的对象，尤其是列表。
- 迭代器可用于：
  - 对集合中的每个项执行操作。
  - 枚举自定义集合。
  - 扩展 LINQ 或其他库。
  - 创建数据管道，以便数据通过迭代器方法在管道中有效流动。

4.2 使用foreach进行循环访问

- 使用 foreach 关键字来枚举集合；
- foreach 依赖2 个泛型接口IEnumerable<T> 和 IEnumerator<T>，才能生成循环访问集合所需的代码。

```
1 //foreach进行循环访问的样例
2 foreach (var item in collection)
3 {
4     Console.WriteLine(item?.ToString());
5 }
```


4.3 异步方法

- 如果序列是 System.Collections.Generic.IEnumerable<T>，则使用 foreach。
- 如果序列是 System.Collections.Generic.IAsyncEnumerable<T>，则使用 await foreach。

```
await foreach (var item in asyncSequence)
{
    Console.WriteLine(item?.ToString());
}
```


4.4 使用迭代器方法的枚举源

- C#具有能够生成创建枚举源的功能，该方法称为迭代器方法，它使用yield return 语句。
- 注意：迭代器方法有一个重要限制：在同一方法中不能同时使用 return 语句和 yield return 语句。

```
1  public IEnumerable<int> GetSetsOfNumbers()
2  {
3      int index = 0;
4      while (index < 10)
5          yield return index++;
6
7      yield return 50;
8
9      index = 100;
10     while (index < 110)
11         yield return index++;
12 }
```


谢谢！