# 基本语法

# C#中out和ref之间的区别

ref（C# 参考）

ref 关键字使参数按引用传递。其效果是，当控制权传递回调用方法时，在方法中对参数的任何更改都将反映在该变量中。若要使用 ref 参数，则方法定义和调用方法都必须显式使用 ref 关键字。

例如：

class RefExample

{

static void Method(ref int i)

{

i = 44;

}

static void Main()

{

int val = 0;

Method(ref val);

// val is now 44

}

}

传递到 ref 参数的参数必须最先初始化。这与 out 不同，后者的参数在传递之前不需要显式初始化。

尽管 ref 和 out 在运行时的处理方式不同，但在编译时的处理方式相同。因此，如果一个方法采用 ref 参数，而另一个方法采用 out 参数，则无法重载这两个方法。例如，从编译的角度来看，以下代码中的两个方法是完全相同的，因此将不会编译以下代码：

class CS0663\_Example

{

// Compiler error CS0663: "cannot define overloaded

// methods that differ only on ref and out".

public void SampleMethod(ref int i) { }

public void SampleMethod(out int i) { }

}

但是，如果一个方法采用 ref 或 out 参数，而另一个方法不采用这两个参数，则可以进行重载，如下例所示：

class RefOutOverloadExample

{

public void SampleMethod(int i) { }

public void SampleMethod(ref int i) { }

}

out（C# 参考）

out 关键字会导致参数通过引用来传递。这与 ref 关键字类似，不同之处在于 ref 要求变量必须在传递之前进行初始化。若要使用 out 参数，方法定义和调用方法都必须显式使用out 关键字。

class OutExample

{

static void Method(out int i)

{

i = 44;

}

static void Main()

{

int value;

Method(out value);

// value is now 44

}

}

尽管作为 out 参数传递的变量不必在传递之前进行初始化，但需要调用方法以便在方法返回之前赋值。

区别：

ref和out的区别在C# 中，既可以通过值也可以通过引用传递参数。通过引用传递参数允许函数成员更改参数的值，并保持该更改。若要通过引用传递参数， 可使用ref或out关键字。ref和out这两个关键字都能够提供相似的功效，其作用也很像C中的指针变量。它们的区别是：

1、使用ref型参数时，传入的参数必须先被初始化。对out而言，必须在方法中对其完成初始化。

2、使用ref和out时，在方法的参数和执行方法时，都要加Ref或Out关键字。以满足匹配。

3、out适合用在需要retrun多个返回值的地方，而ref则用在需要被调用的方法修改调用者的引用的时候。

out

方法参数上的 out 方法参数关键字使方法引用传递到方法的同一个变量。当控制传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。

当希望方法返回多个值时，声明 out 方法非常有用。使用 out 参数的方法仍然可以返回一个值。一个方法可以有一个以上的 out 参数。

若要使用 out 参数，必须将参数作为 out 参数显式传递到方法。out 参数的值不会传递到 out 参数。

不必初始化作为 out 参数传递的变量。然而，必须在方法返回之前为 out 参数赋值。

属性不是变量，不能作为 out 参数传递。

ref是有进有出，而out是只出不进。  
（之前看过忘了，转载于https://www.cnblogs.com/sunliyuan/p/5999045.html）

# c# yield关键字和Unity协程

方法中yield Return返回一个迭代器，所以其返回的一定是IEnumerator。

Yield类似一个暂停，迭代运行到 yield return 时，会直接返回，然后在下次迭代时，再从上次暂停的地方继续运行。

**Yield Break : 结束迭代**

在迭代器块中用于向枚举数对象提供值或发出迭代结束信号。

**yield注意事项**：

**1 不能将 yield return 语句置于 try-catch 块中。 可将 yield return 语句置于 try-finally 语句的 try 块中。**

**2 可将 yield break 语句置于 try 块或 catch 块中，但不能将其置于 finally 块中。**

**3 如果 foreach 主体（在迭代器方法之外）引发异常，则将执行迭代器方法中的 finally 块。**

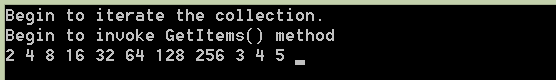
**4 匿名方法。 有关详细信息，请参阅匿名方法。**

**5 包含不安全的块的方法。 有关详细信息，请参阅unsafe。**

**实例代码：**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **namespace** ConsoleApplication2
7. {
8. **class** Program
9. {
10. **static** **void** Main(string[] args)
11. {
12. //这儿调用了方法。
13. var test = Power(2, 8, "");
14. Console.WriteLine("Begin to iterate the collection.");
15. // 返回2的各个次方，如2,4,8,16,32等
16. foreach (**int** i in Power(2, 8, ""))
17. {
18. Console.Write("{0} ", i);
19. }
20. Console.ReadKey();
21. }
22. **public** **static** IEnumerable<**int**> Power(**int** number, **int** exponent, string s)
23. {
24. **int** result = 1;
25. **if** (string.IsNullOrEmpty(s))
26. {
27. //throw new Exception("这是一个异常");
28. Console.WriteLine("Begin to invoke GetItems() method");
29. }
31. **for** (**int** i = 0; i < exponent; i++)
32. {
33. result = result \* number;
34. yield **return** result;
35. }
36. yield **return** 3;
37. yield **return** 4;
38. yield **return** 5;
39. }
40. }
41. }

结果：

  
摘自：<http://blog.csdn.net/cbbbc/article/details/50232929>

## **Unity中的运用**

协程coroutines yield return  
Unity coroutines   
例子功能：   
不同阶段不同行为的GameObject。   
阶段1，它移动到特定坐标后打印“我到了”   
阶段2，过3秒后再打印“完成！”

public class CoroutinesExample : MonoBehaviour

{

public float smoothing = 1f;

public Transform target;

void Start ()

{

StartCoroutine(MyCoroutine(target));

}

// 注意这个函数返回值是IEnumerator，必须

IEnumerator MyCoroutine (Transform target)

{

// 处理第一阶段

// 和目标大于0.05就按照smoothing \* Time.deltaTime移动一段距离

while(Vector3.Distance(transform.position, target.position) > 0.05f)

{

transform.position = Vector3.Lerp(transform.position, target.position, smoothing \* Time.deltaTime);

yield return null;// 这里暂停后返回，等待下帧执行机会

}

print("我到了");

// 处理第二阶段

yield return new WaitForSeconds(3f);

print("完成！");

}

}

unity中的yield   
yield 后面可以跟的表达式：   
a) return null - 等下个Update之后恢复   
b) return new WaitForEndOfFrame() - 等下个OnGUI之后恢复   
c) return new WaitForFixedUpdate() - 等下个FixedUpdate之后恢复，有可能一帧内多次执行   
d) return new WaitForSeconds(2) - 2秒后，等下个Update之后恢复   
e) return new WWW(url) - Web请求完成了，Update之后恢复   
f) return StartCorourtine() - 新的协成完成了，Update之后恢复   
g) break -退出协程   
h) return Application.LoadLevelAsync(levelName) - load level 完成，异步加载场景   
i) return Resources.UnloadUnusedAssets(); // unload 完成

注意其中 WaitForSeconds()受Time.timeScale影响，当Time.timeScale = 0f 时，yield return new WaitForSecond(x) 将不会恢复。

# C# set get访问器

set 访问器与返回 void 的方法类似。它使用称为 value 的隐式参数，此参数的类型是属性的类型。

Get访问器的返回有属性类型一样。

* 只带有 get 访问器的属性称为只读属性。无法对只读属性赋值。
* 只带有 set 访问器的属性称为只写属性。只写属性除作为赋值的目标外，无法对其进行引用。
* 同时带有 get 和 set 访问器的属性为读写属性。

访问器的权限和字段本身的权限一样，可以单独给其中一个访问器设置权限如：private set，但是新设置的权限大小 不能大于 字段的权限大小。

# Operator运算符重载与Implicit隐式类型转换

1. class Person
3. {
4. public int Age { get; set; }
5. public string Name { get; set; }
7. public static int operator -(Person p1, Person p2)//int为想减结果类型，-表示对-进行运算符重载
8. //参数为待计算的两个值
9. {
10. return p1.Age - p2.Age;
11. }
12. //当重载>后，要求必须重载<
13. public static bool operator >(Person p1, Person p2)
14. {
15. return p1.Age > p2.Age;
16. }
18. public static bool operator <(Person p1, Person p2)
19. {
20. return p1.Age < p2.Age;
21. }
22. //将string 转换成Person ，对象的隐式转化，这里这里返回bool，在if时就可以直接用了
23. public static implicit  operator Person(string s)
24. {
25. Person p = new Person();
26. p.Name = s;
27. return p;
28. }
29. //将Person 转换成string
30. public static implicit operator string(Person p)
31. {
32. return p.Name;
33. }
35. //当重载==后，要求必须重载!=
37. public static bool operator ==(Person p1, Person p2)
38. {
39. return p1.Name == p2.Name && p1.Age == p2.Age;
40. }
42. public static bool operator !=(Person p1, Person p2)
43. {
44. return !(p1 == p2);
45. }
46. }