6.Yield 同等实现

在昨天，我们学习了，如何使用 Yield。今天我们来深入学习 Yield 关键字。

我们贴出示例代码

```
/************************************************************************
 * http://sikiedu.com liangxie
 ************************************************************************/

using System.Collections;
using UnityEngine;

namespace UniRxLesson
{
    public class YieldDeepExample : MonoBehaviour
    {
        private void Start()
        {
            var enumeratorTest = new IEnumeratorTest();

            foreach(var item in enumeratorTest)
            {
                Debug.Log(item);
            }
        }

        class IEnumeratorTest
        {
            public IEnumerator GetEnumerator()
            {
                yield return 1;
                yield return 2;
                yield return "枚举器";
            }
        }
```

```
    }
}
```

输出结果为

```
1
2
"枚举器"
```

注意 IEnumeratorTest 没有实现 IEnumerator 接口，但是还是可以用 foreach 来迭代。这是因为，编译器只关心是否实现了 GetEnumerator() 方法。

很简单。

yield 解释说明

我们把包含 yield 语句的方法或属性称为迭代块。如上面代码代码中的：

```
public IEnumerator GetEnumerator()
{
    yield return 1;
    yield return 2;
    yield return 枚举器;
}
```

这个语句块在编译时将会编译成一个 yield 类型，其中包含一个状态机。yield 类型实现 IEnumerator 和 IDisposable 接口属性和方法。

上面的

```
class IEnumeratorTest
{
    public IEnumerator GetEnumerator()
```

```
    {
        yield return 1;
        yield return 2;
        yield return "枚举器";
    }
}
```

则会编译成如下类似的代码。

```
public class IEnuemratorTest
{
    public IEnumerator GetEnumerator()
    {
        return new Enumerator(0);
    }

    public class Enumerator:IEnumerator<object>,IEnumerator,IDisposable
    {
        private int state;
        private object current;
        public Enumerator(int state)
        {
            this.state = state;
        }

        public object Current
        {
            get
            {
                return current;
            }
        }

        public void Dispose()
        {
        }

        public bool MoveNext()
        {
```

```
            switch(state)
            {
                case 0:
                    state++; // 改变当前迭代诶值为下一个元素位置
                    current = 1; // 当前迭代位置的对象
                    return true;
                case 1:
                    state++; // 改变当前迭代位置为下一个元素位置
                    current = 2; // 当前迭代位置的对象
                    return true;
                case 2:
                    state++; // 改变当前迭代位置为下一个元素位置
                    current = "枚举器"; // 当前迭代位置的对象
                    return true;
            }
            return false;
        }

        public void Reset()
        {
        }
    }
}
```

枚举器其实就是通过每次调用 MoveNext() 方法，来改变集合中当前元素位置，来一个个遍历元素。类似通过更改下标来获取元素。yield 关键字其实是实现 IEnumerator 等接口如 MoveNext() 方法的编译器自动编译的关键字。当然你也可以自己实现 MoveNext() 等方法，如果你不怕麻烦。

yield 的介绍就到这里。