1.Yield的用法

Yield是和IEnumerable配合使用的

Yield主要有两个特性：

第一个是只有真正调用的时候才会去获取

第二个是Yield return后下一次movenext如果不为false的话则继续从上次yield的地方开始向下执行代码。（这个地方之所以能够继续从上次的 地方开始是用了go to + state + switch=>就形成了我们的状态机）

这个东西感觉用处还是很大的，可以不必从头到尾开始执行代码，可以根据具体情况，往下接着执行对应的代码。

2.迭代器

迭代器的定义：就是对于不同的集合提供一个通用 的访问方法，而且还可以实现按需获取

扩展的话，就是实现了IEnumerable或者说IEnumerator GetEnumerator()方法的类

进一步IEnumerator 里面就是实现下面的方法。

这样的话就都可以用ForEach进行遍历了

```
namespace System.Collections
{
    public interface IEnumerator
    {
        //
        // 摘要:
        //     获取集合中位于枚举数当前位置的元素。
        //
        // 返回结果:
        //     集合中位于枚举数当前位置的元素。
        object Current { get; }

        //
        // 摘要:
        //     将枚举数推进到集合的下一个元素。
        //
        // 返回结果:
        //     如果枚举数已成功地推进到下一个元素，则为 true；如果枚举数传递到集合的末尾，
        //
        // 异常:
        //   T:System.InvalidOperationException:
        //     创建枚举器后，已修改该集合。
        bool MoveNext();
        //
        // 摘要:
        //     将枚举数设置为其初始位置，该位置位于集合中第一个元素之前。
        //
        // 异常:
        //   T:System.InvalidOperationException:
        //     创建枚举器后，已修改该集合。
        void Reset();
    }
}
```

第三：多线程后续相关

1.多线程的异常。

正常情况下面多线程的异常是不能抓到的，因为是异步的。可以用 下面的方法来抓取多线程异常

（1）.通过WaitAll加上AggregateException类型异常来获取
（2）因为waitall会出现卡界面的情况，不太理想。正常来说线程里面是不能出现异常的，所以可以应该在线程方法里面直接写try catch保证线程里面不出错。且把错误在内部处理。

2.线程取消

即是多个线程同时执行，一个失败后希望其他的也一起停下来。
（1）task是外部无法中止的，Thread.Abort不靠谱，因为线程是OS的资源，，无法掌控什么时候取消
（2） cts的方式

```
////多个线程并发，某个失败后，希望通知别的线程，都停下来
////task是外部无法中止，Thread.Abort不靠谱，因为线程是OS的资源，无法掌控啥时候取消
////线程自己停止自己--公共的访问变量--修改它---线程不断的检测它(延迟少不了)
////CancellationTokenSource去标志任务是否取消 Cancel取消 IsCancellationRequested 是否已经取消了
////Token 启动Task的时候传入，那么如果Cancel了，这个任务会放弃启动，抛出一个异常
CancellationTokenSource cts = new CancellationTokenSource();//bool值 //bool flag = true;
3. //多线程中临时变量和全局变量
//即是如果用的全部变量大家都能同时访问到的话那么就有可能出现问题
4. #region 线程安全 lock
//共有变量：都能访问局部变量/全局变量/数据库的一个值/硬盘文件
//线程内部不共享的是安全

//lock 解决，因为只有一个线程可以进去，没有并发，所以解决了问题 但是牺牲了性能，所以要尽量缩小lock的范围
//不要冲突--数据拆分，避免冲突

//安全队列 ConcurrentQueue 一个线程去完成操作
```