协同程序

类似于 多线程的概念。

协程和线程的区别：

一个多线程的程序，可以同时运行多个线程。而协程呢，在某个时刻，只有一个协程在运行。

线程由cpu调度，协程由代码调度。

创建协程，并运行：

```
# 定义协程
testAdd = coroutine.create(
        function(a,b)
                print(a+b)
        end
)
# 启动协程
# 原来暂停-》执行，原来执行-》暂停
coroutine.resume(testAdd, 1,2)
```

wrap

```

co = coroutine.wrap(
        function(a)
                print("参数值是："..a)

        end
)
co(2)
```

启动、停止

```
testAdd = coroutine.create(
        function(a,b)
                print("执行--子方法",a+b)
                coroutine.yield();
                print("执行--子方法",a-b)
        end
)

coroutine.resume(testAdd, 1,7)
```

```
print("执行主方法")
coroutine.resume(testAdd)
```

返回值

```
testAdd = coroutine.create(
        function(a,b)

                print("协程执行",a+b)
                coroutine.yield()

                return a+b,a-b
        end
)

r1,r2,r3 = coroutine.resume(testAdd, 1,7)
print("返回值：",r1,r2,r3)

r1,r2,r3 = coroutine.resume(testAdd, 1,7)
print("重新执行，返回值：",r1,r2,r3)
```

协程状态

```
testAdd = coroutine.create(
        function(a,b)
                print("运行中 协程状态：",coroutine.status(testAdd))
                coroutine.yield()
                return a+b,a-b
        end
)

print("刚定义好的协程状态：",coroutine.status(testAdd))

r1 = coroutine.resume(testAdd,1,4)
print("启动协程结果：",r1)

print("最终的 协程状态：",coroutine.status(testAdd))

print("yield后 协程状态：",coroutine.status(testAdd))

r1 = coroutine.resume(testAdd,1,4)
print("二启动协程结果：",r1)

print("二最终的 协程状态：",coroutine.status(testAdd))
```

```
r1 = coroutine.resume(testAdd,1,4)
print("三启动协程结果：",r1)
```

结果：

```
刚定义好的协程状态：    suspended
运行中 协程状态：       running
启动协程结果：  true
最终的 协程状态：       suspended
yield后 协程状态：      suspended
二启动协程结果：        true
二最终的 协程状态：     dead
三启动协程结果：        false
```

协程协作

协程唯一标识

```
testAdd = coroutine.create(
        function(a,b)
                print(coroutine.running)
                print("1")
        end
)

coroutine.resume(testAdd,1,1)
```

协程内部和外部协作的例子：

```
-- 协程内外部协作的例子
function foo(a)
        print("foo 参数：",a)
        return coroutine.yield(a*2)

end

co = coroutine.create(
        function(a,b)
                print("第一次启动协程，参数：",a,b)
                local r = foo(a+1)
                print("第二次启动协程，参数",r)
```

```
        local x,y = coroutine.yield(a+b,a-b)
        print("第三次启动协程，参数",x,y)

        return b,"协程结束啦"

    end
)

print("主程序：",coroutine.resume(co,1,5))
print("----分隔符---")
print("主程序：",coroutine.resume(co,"r"))
print("----分隔符---")
print("主程序：",coroutine.resume(co,"x","y"))

print("----分隔符---")
print("主程序：",coroutine.resume(co))
```

第一次resume，传入的参数是 function的参数。

第一次yield的参数，是第一次resume的返回值。

第二次resume的参数，是第一次yield的 返回值。

生产者消费者问题

思路：

1。生产者生产完 产品，（自己停下来），等待消费者消费。

2。消费者消费完产品，（自己停下来），等待生产者生产。

```
-- 生产者和消费者
function productor()
    -- 定义生产的商品，用数字来替代
    local i = 0
    while i<100
    do
        i = i+1
        print("生产了：",i)
        -- 通过协程实现
        coroutine.yield(i)

    end


end
```

```
function consumer()
        while true
        do
                -- 从生产者获取产品
                local status,result = coroutine.resume(po)
                print("消费了：",result)
                if (result == 99) then
                        break
                end
        end


end

-- 程序开始
po = coroutine.create(productor)
consumer()
```