

7.2 is 和 == 的区别

== 其实内部是 \_\_equals\_\_方法, 比较的是值

is 是 id() 方法，比较的是id值

但是出现 :

a=[1,2,3,4]

b=[1,2,3,4]

这种赋值的时候，a和b的id就不一样了

而 b=a 这种引用的时候，a和b的id就是一样了

**\*\*\*\*重要**：：在Python中有一个 intern 机制，当出现 小整数 或者小字符串的时候，

Python会 建立一个全局为 小整数的对象，下一次再用小整数的时候，就会指向这个全局

为小整数的对象， 即他们的 id 值是相等的， 并不会再去重新生成一个，这是Python内

部的优化机制

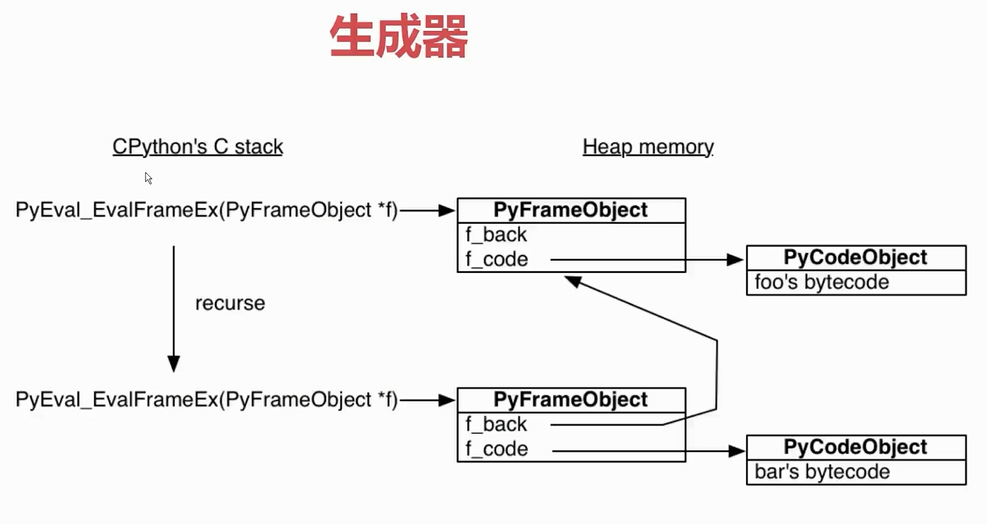
即： a=1

b=1

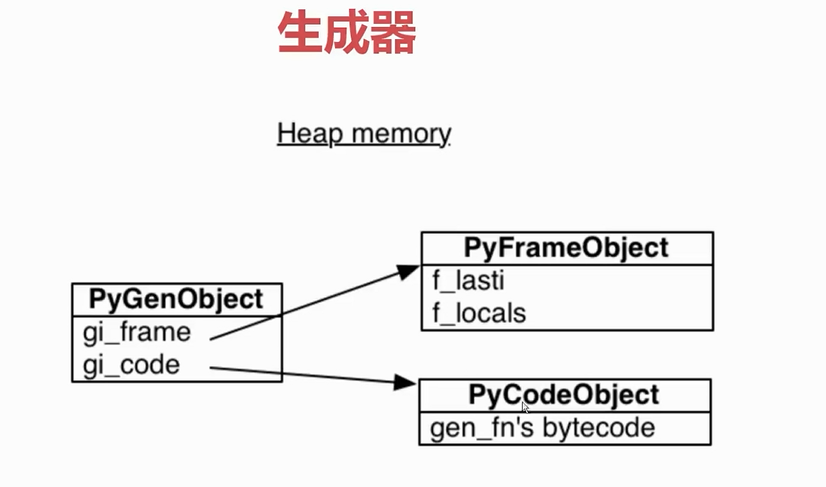
id(a) 和 id (b) 是相等的

**------------------------------------------------------------------------------------------**

**生成器执行过程**

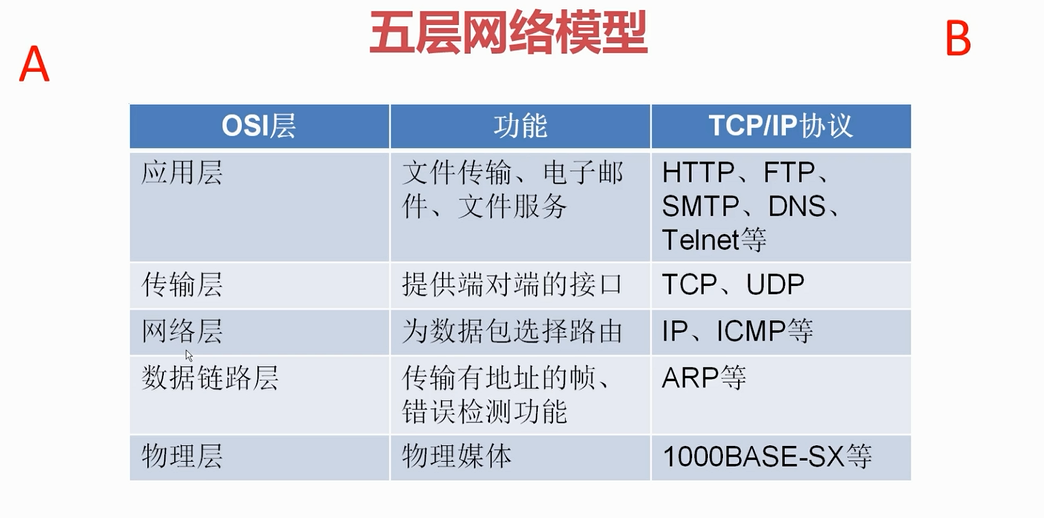


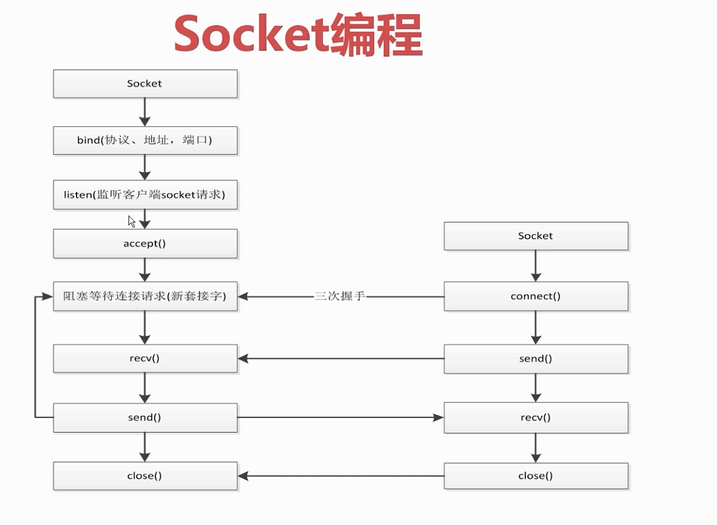
**生成器对象**

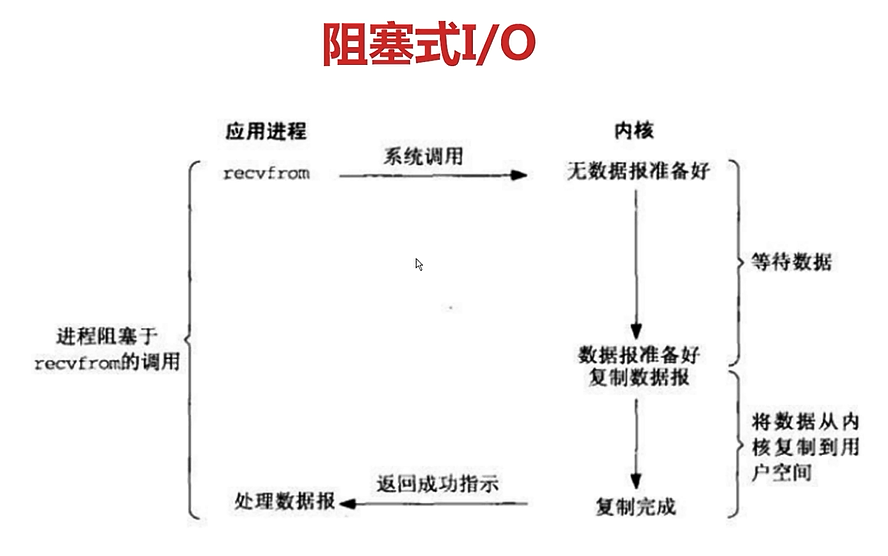


f\_lasti 指向最近执行的代码

每一次yield都会暂停，记录最近执行代码的字节码的位置和locals变量

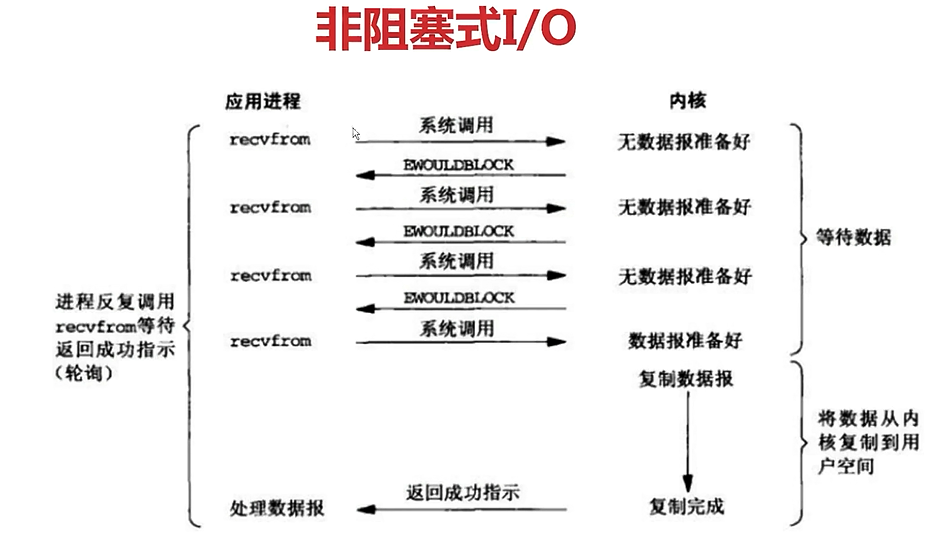






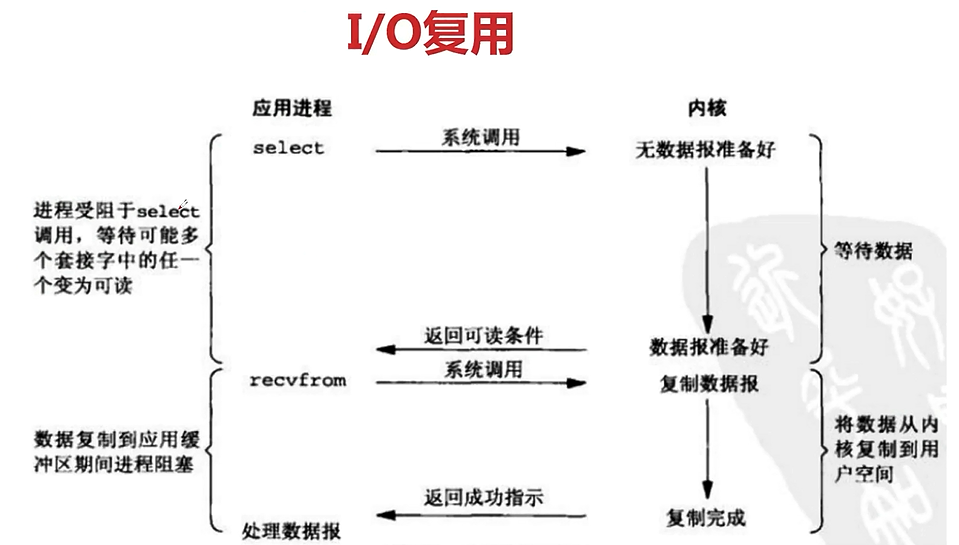
极大地浪费cpu资源，，

但并不会消耗cpu（cpu没有运行）



cpu没有继续等待，比如连接的时候（connect 三次握手）但是在不停的询问连接是否建立好，需要while循环不停的去检查状态，耗费cpu

但是当做计算任务或者再次发起其他的连接请求的时候就很有用了（即不需要检查当前的连接）

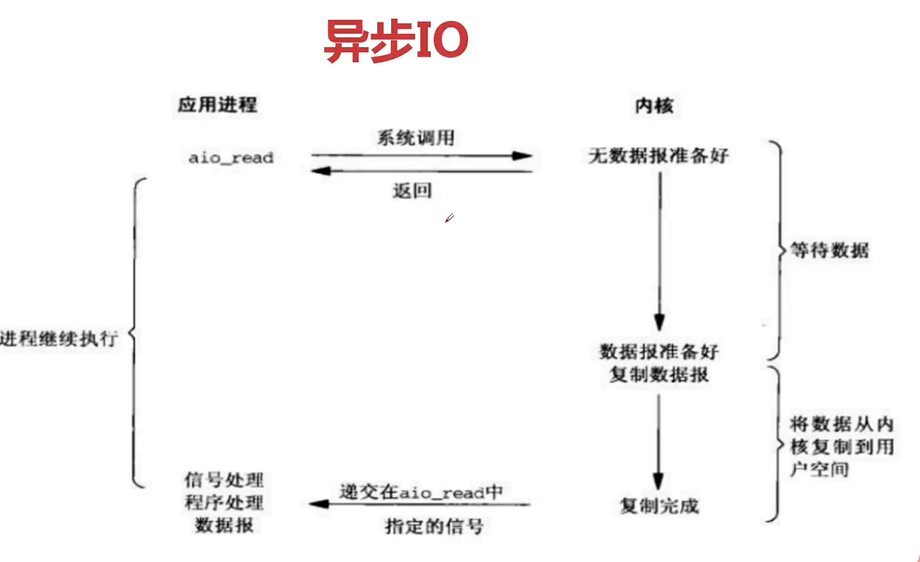


select 也是一个阻塞式的方法，但是和while for循环有很大的区别，可以同时监听多个状态，一旦有一个状态发生变化，立马可以去处理

当数据准备好之后 返回一个命令，然后进行数据复制处理

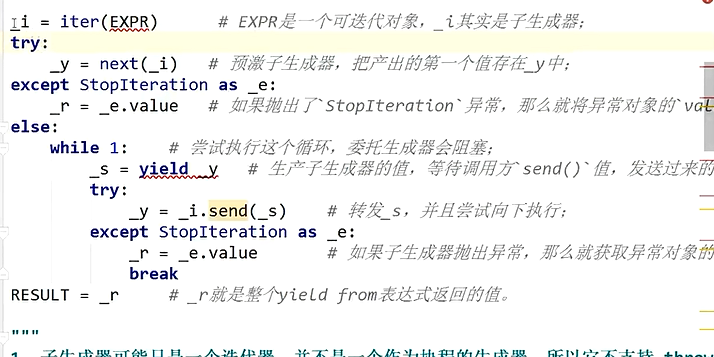


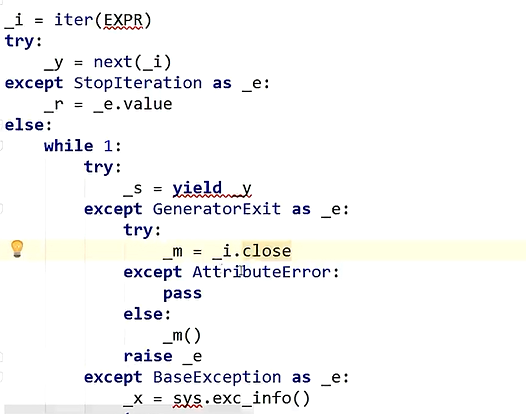
基于信号来的，操作系统主动给处理程序发送信号，用的很少

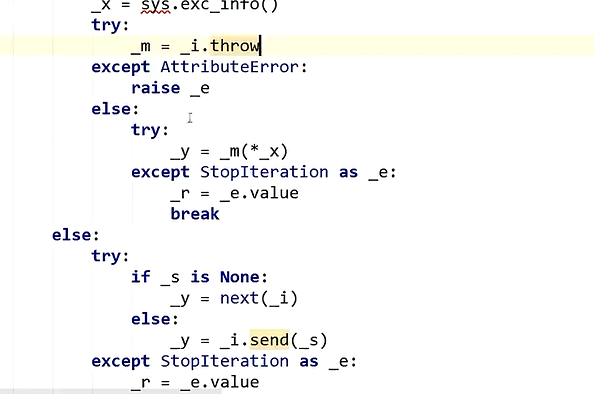


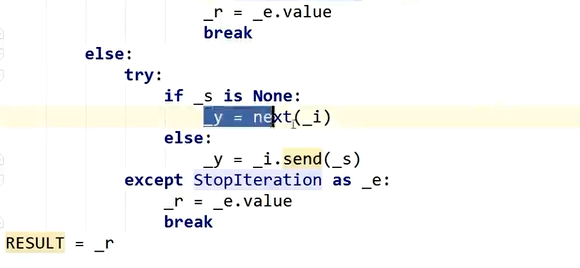
这才是真正的异步IO，当数据复制完后才发送命令，但是相对于IO复用来说，提升并不是很明显，因此现在的异步IO基本上大多数都是 IO复用

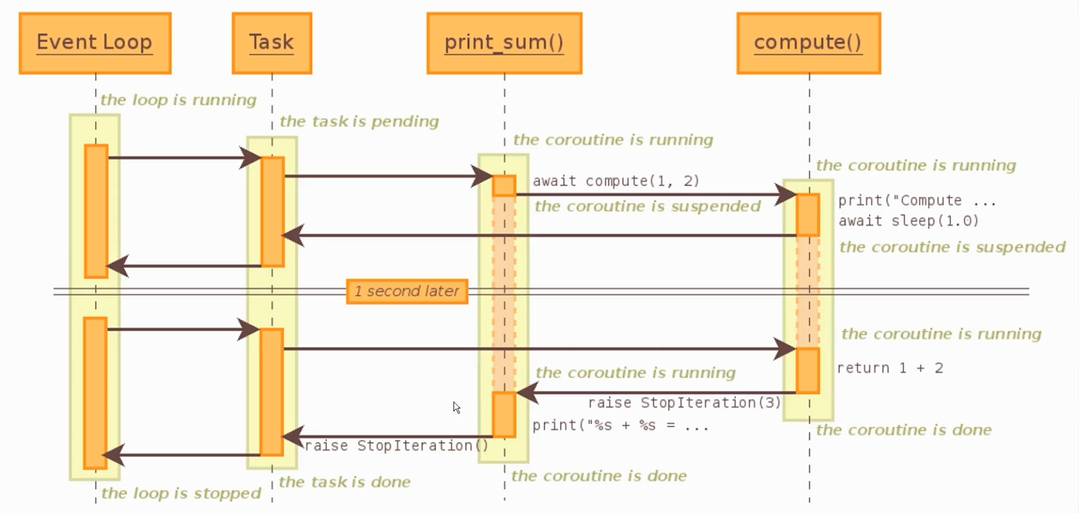
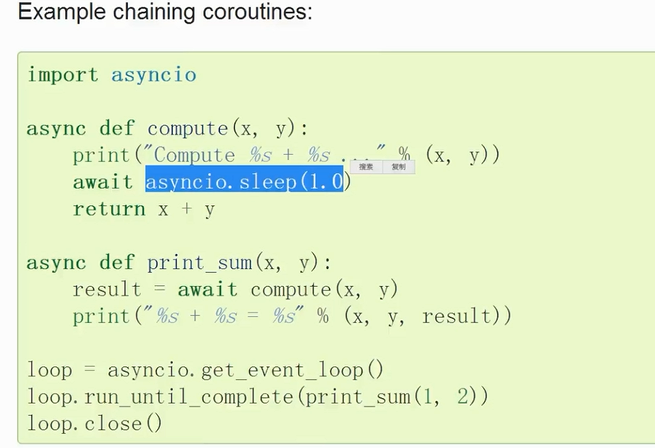
yield from 背后处理的异常











子协程调用原理 时序图

EventLoop -> Task ->print\_sum() [相当于委托生成器]-> compute() ->Task ->EventLoop

EventLoop ->Task ->compute() -> print\_sum() ->Task ->EventLoop