# 随堂笔记-并发编程

## 主要内容

* 什么是并发编程
* 并发编程的实现方式和手段
* 并发编程的操作实例
* 多线程并发编程
* 多进程并发编程
* 协程并发编程
* 扩展：异步IO、异步网络

## 课程内容

### 并发编程的意义

结合现实生活：生活中有一个事情~备课、吃饭、视频

一个人在同一时间，同时执行多个事情的过程：并发的过程

反应到代码中：顺序代码：按照代码解释顺序逐行执行。

上一行代码没有执行完成，下一行代码等待[阻塞]

res = input()

print(res)

顺序代码：存在什么样的优点，有什么样的缺陷

优点：思路清晰，按照步骤执行

缺陷：同时只能执行一个任务

**一段时间只做一件事情**

**案例：火车站卖票**

**10窗口**

**10000人排队**

**1号窗口售票[1000]**

**...**

**顺序情况下：即使安排多个窗口售票，售票总时间并没有发生变化**

**并发情况下：多个窗口同时售票，售票总时间 原来的总时间/窗口数量-> 提高执行效率**

### 什么是并发

所谓并发：**表面上**是同时执行，并发是相对于**顺序执行**的概念

**串行**：按照步骤，一步一步执行，前面的步骤没有执行完成，后面的步骤等待

**并发**：在很短的时间之类，多个任务切换执行，在一个时间段~体现出来一种几个任务同时执行的表象

**并行**：几个任务同时执行

|  |
| --- |
| **小总结：**  **什么是并发？**  **什么是并发编程？**  **串行、并行、并发有什么意义和区别？**  **同步和异步分别是什么意思，分别怎么对应同时/不同时？**  **简单了解什么是进程？什么是多进程？**  **简单了解什么是线程？什么是多线程？** |

### PYTHON中的并发编程支持

* 多线程并发机制
  + 单进程多线程：如果多个执行单元需要频繁访问公共数据的时候【多线程并发】
* 多进程并发机制
  + 多进程单线程：如果多个执行单元执行不同任务同时访问不同的独立数据【多进程并发】
* 协程并发机制
  + 单进程单线程：并发：通过程序代码的方式，完成模拟CPU时间片并发任务的处理机制

### 多线程并发编程

PYTHON的版本

python1.x[过时]

python2.x[了解]

python3.x[推荐]

python2中提供了两种多线程并发的模块

thread：提供给并发编程专业开发人员使用的底层模块

threading：提供给应用程序开发人员并发底层开发模块

python3中对于多线程并发的模块

\_thread：约定私有的，表示非专业人员不要触碰。

threading：有限推荐使用的并发编程模块

\_thread不推荐使用

|  |
| --- |
| *''' AUTHOR: DAMU 牟文斌 VERSION: V1.0.000 DESC: 多线程底层并发\_thread '''* **import** \_thread, time   **def** test(name):  *'''并发执行测试'''* **for** i **in** range(10):  time.sleep(0.1)  print(name, **">>>>>>"**, i)   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  *# 创建两个子线程* \_thread.start\_new\_thread(test, (**'test1'**, ))  \_thread.start\_new\_thread(test, (**'test2'**, ))   *#* time.sleep(2)  print(\_\_name\_\_, **"主线程退出"**) |
| *备注：因为\_thread模块可以实现并发，但是实现过程中，有大量约定的底层逻辑规范*  *同时需要控制大量线程状态来完成线程的启动、执行、暂停(阻塞)、死亡等等状态，底层技术面较多，导致开发人员大量的精力放在技术上而不是业务上，造成了开发效率的下降，所以不推荐使用。* |

threading：推荐的并发方案

Thread：类型，专门用于并发编程的多线程支持类型

可以支持面向过程中的多线程并发[函数式多线程]

可以支持面向对象的多线程并发[面向对象编程]

Event 事件；控制线程之间的通信

Condition 条件；控制线程之间的通信

Lock/RLock 锁；控制线程之间的通信

..

|  |
| --- |
| *''' AUTHOR: DAMU 牟文斌 VERSION: V1.0.000 DESC: 多线程并发编程 '''* **import** threading, time  count\_ticket = 10   **def** sale\_ticket():  *'''售票窗口'''* **global** count\_ticket   **while** count\_ticket > 0:  print(threading.current\_thread().name, **"售出一张票"**, count\_ticket)  count\_ticket -= 1  time.sleep(1)   **if** \_\_name\_\_ == **"\_\_main\_\_"**:  **'''多线程并发测试代码'''** *# 创建两个窗口:创建两个可以同时执行的子线程对象* t1 = threading.Thread(target=sale\_ticket, name=**'窗口1'**)  t2 = threading.Thread(target=sale\_ticket, name=**'窗口2'**)   *# 启动线程* t1.start()  t2.start()   print(**"before time sleep-->"**, t1.is\_alive())  time.sleep(10)  print(**"after time sleep-->"**, t1.is\_alive())  *# t1.start() # RuntimeError: threads can only be started once* |
| *备注：一个线程，启动之后，不论是正在运行状态还是已经运行结束状态(死亡状态)该线程都不能被重复启动【一个线程在一个执行过程中，只能启动一次】* |

多个线程访问数据，很有可能会出现数据访问冲突的问题

多线程并发编程中[不论任何高级语言]，在处理多线程并发访问数据冲突的问题时都是通过线程锁对数据实施保护的

python中提供了两种比较有用的锁

同步锁|互斥锁：Lock

可重用锁：RLock

锁对象：lock = threading.Lock()/RLock()

上锁、锁定【术语：获取锁】：lock.acquire() -> bool

解锁、开锁【术语：释放锁】：lock.release()

锁的使用~并不是对数据/函数/方法直接锁定即可，锁的范围很重要

锁定的范围过大：会导致多线程程序出现~单线程执行->**效率并没有提升**

锁定的范围过小：对于锁定的数据并没有产生有效的保护

锁定范围：一般情况针对可能冲突的数据[修改-关联的查询]锁定到一起即可。

注意：一定要注意，如果多线程访问时，多个数据使用多个不同的锁~互相锁定数据的情况

死锁：程序中的一种BUG【BUG不是错误~运行过程中触发了某些条件产生意料之外的情况】

死锁会直接导致程序阻塞。

线程的状态：运行时Runtime

线程—独占模式—join()

线程-守护模式-daemon=False/True 注意使用方式

线程-存货状态-is\_alive()

..

线程的通信

多个线程之间如果出现如下场景，就需要线程之间进行通信

A线程执行过程中，需要依赖B线程提供的数据-> 必须要求B线程完成数据的准备A线程才能执行后续的代码，此时就需要A线程和B线程之间实施通信操作完成多个线程协同运行。

线程通信使用的内置类型

threading.Lock/RLock 最基本的通信

threading.Event 线程之间互相调用-阻塞运行之间的调度

threading.Condition 条件对象

晚自习任务：

TASK1：完成端对端的无缝聊天[TCP网络编程 服务端-客户端]

主线程-服务端/客户端

子线程1：发送消息

子线程2：接受消息

TASK2：扩展：完成聊天室[UDP编程]

TASK3：梳理资料

什么是并发

什么是并发编程

python中对并发编程的支持

什么是线程，什么是多线程编程

什么是进程，什么是多进程编程

python中的\_thread模块和threading模块

模块的作用

模块中常见的类型/方法/函数

\_thread实现的简单多线程，和注意事项

threading实现的面向函数的多线程

多线程购票程序[多个窗口同时售票]

多线程-数据保护-线程锁[Lock|RLock]

多线程-线程通信-事件[Event]-买早餐[核心：两个线程之间的数据依赖]

多线程-线程通信-条件[Condition]-生产者消费者[**TODO**]

多线程-数据容器-队列[Queue]-生产者消费者[**TODO**]

扩展：信号量对象..