并发编程

- 概述
  - 非并发
    - 程序由单个步骤序列构成
    - 包含独立子任务的程序执行性能低
  - 并发
    - 异步 / 高效
    - 分解子任务 / 简化流程与逻辑
  - 进程 process
    - 一个程序的执行实例
    - 每个进程都有自己的地址空间 / 内存 / 数据栈 及辅助数据
  - 线程 thread
    - 同一个进程内,可被并行激活的控制流
    - 共享相同的上下文(空间地址, 数据结构)
    - 特点
      - 便于信息共享和通信
      - 线程访问顺序差异会导致结果不一致(条件 race condition)
  - Python GIL 全局解释器锁
    - python 代码 由虚拟机(解释主循环) 控制
    - 主循环只能有一个控制线程执行
- 多线程
  - _thread
    - 特点
      - 没有控制进程结束机制
      - 只有一个同步原语(锁)
      - 功能少于threading模块
    - .start_new_thread(function, args) 开始线程
  - threading模块
    - .Thread 线程类
      - 构造
        - .Thread(target=目标函数, args=(参数,))
        - 自定义Thread派生类, 重写run 方法逻辑
        - .start() 启动线程
        - .join() 要求主线程等待
        - .name 线程名称
    - .current_thread() 获取当前线程
    - threading.Lock 同步原语: 锁
      - .acquire()获得
      - .release() 释放
      - 支持上下文操作 with lock:
- 队列

        - queue 模块
        - queue FIFO
            - .Queue()构造实例
            - put(item, block=True, timeout=None)放入数据项
            - .get(block=True, timeout=None) 获取数据项
            - .task_down 声明当前队列任务处理完毕
            - .join () 队列所有项处理完毕前阻塞
        - LifoQueue LIFO
        - PriorityQueue 优先队列
- multiprocessing模块
    - 充分运用多核\多CPU的计算能力, 适用于计算密集型的任务
- concurrent.futures 模块
    - ThreadPoolExecutor
    - ProcessPoolExecutor