School of Economics ang Management, Beihang University

现代程序设计技术

赵吉昌
jichang@buaa.edu.cn

本周内容

- 面向对象编程
  - 异步IO
  - 协程

IO模型

- 同步IO
  - 在IO过程中当前线程被挂起，当前线程其他需要CPU计算的代码无法执行
  - 多线程可解决该问题
    - 计算和IO任务可以由不同的线程负责
    - 但会带来线程创建、切换的成本，而且线程数不能无上限地增加
- 异步IO
  - 当前线程只发出IO指令，但不等待其执行结束，而是先执行其他代码，避免线程因IO操作而阻塞

异步IO

- 事件驱动模型
  - 一种编程范式，程序执行流由外部事件决定
  - 包含一个事件循环，当外部事件发生时使用回调机制来触发相应的处理
  - 可能的实现机制
    - 每收到一个请求，创建一个新的进程来处理该请求；
    - 每收到一个请求，创建一个新的线程来处理该请求；
    - 每收到一个请求，放入一个事件列表让主进程通过非阻塞IO方式来处理请求
  - 一般场景
    - 当程序中有许多任务，任务之间高度独立（不需要互相通信或等待彼此等），并且在等待事件到来时，某些任务会阻塞

异步IO

- 事件列表模型
  - 主线程不断重复“读取请求-处理请求”这一过程
  - 进行IO操作时相关代码只发出IO请求，不等待IO结果，然后直接结束本轮事件处理，进入下一轮事件处理
  - 当IO操作完成后，将收到IO完成消息，在处理该消息时再获取IO操作结果
  - 在发出IO请求到收到IO完成消息期间，主线程并不阻塞，而是在循环中继续处理其他消息
  - 对于大多数IO密集型的应用程序，使用异步IO将大大提升系统的多任务处理能力

协程

- 协程
  - Coroutine, peusdo-thread, micro-thread
  - 微线程
  - 在一个线程中会有很多函数，一般将这些函数称为子程序，在子程序执行过程中可以中断去执行别的子程序，而别的子程序也可以中断回来继续执行之前的子程序，这个过程就称为协程
    - 执行函数A时，可以随时中断，进而执行函数B，然后中断B并继续执行A，且上述切换是自主可控的
    - 但上述过程并非函数调用（没有调用语句）
  - 表象上类似多线程，但协程本质上只有一个线程在运行

协程

- 协程的优点
  - 无需线程上下文切换的开销，协程避免了无意义的调度，由此可以提高性能
  - 无需原子操作锁定及同步的开销
  - 方便切换控制流，简化编程模型
    - 线程由操作系统调度，而协程则是在程序级别由程序员自己调度
  - 高并发+高扩展性+低成本
    - 一个CPU可以支持上万协程
    - 在高并发场景下的差异会更突出

协程

- 协程的缺点
  - 程序员必须自己承担调度的责任
  - 协程仅能提高IO密集型程序的效率，但对于CPU密集型程序无能为力
  - Python2和Python3中实现有一定差别
    - 所用模块有区别
    - 相关生态还在不断成熟
  - 在CPU密集型程序中要充分发挥CPU利用率需要结合多进程和协程

协程的实现

- 通过yield关键字实现协程
  - 生成器的send()函数
    - 与next()作用类似，但可以发送值给对应的yield表达式
    - 支持外部程序与生成器的交互
  - next(g)就相当于g.send(None)
  - 注意第一次调用next()或send(None)相当于启动生成器，不能使用send()发送一个非None的值
    - 利用装饰器来解决该问题
    - 在装饰器中先调用一次next
  - Demo: ysend.py, dys.py, yc.py, search.py

协程的实现

- 通过greenlet实现协程
  - A “greenlet” is a small independent pseudo-thread
  - When you first switch to it, it starts to run a specified function, which may call other functions, switch out of the greenlet
  - 通过switch交互数据
    - During a switch, an object or an exception is “sent” to the target greenlet; this can be used as a convenient way to pass information between greenlets.
  - Demo: greenletc.py, greenlet_cp.py

协程的实现

- 通过gevent实现协程
  - 基于greenlet
  - spawn构建新协程
  - monkey.pach_all将第三方库标记为IO非阻塞
  - 通过协程池控制协程数目
  - Demo
    - geventc.py
    - gevent_urlhost.py
    - monkey_urllenth.py
    - gevent_port.py
    - geventc_pool.py

协程的实现

- 通过asyncio实现协程
  - python3.4引入
  - 内置异步IO
  - 用asyncio提供的@asyncio.coroutine将任务标记为coroutine类型，然后在coroutine内部用yield from调用另一个coroutine实现异步操作
  - Python3.5开始引入了async和await进一步简化语法
    - 把@asyncio.coroutine替换为async
    - 把yield from替换为await
  - Python3.7进一步变化…

协程的实现

- 通过asyncio实现
  - 注意阅读官方最新的示例
  - 完整的“生态”似乎没有形成
  - Demo
    - asyc.py
    - asyc_url.py
    - asyc_url2.py
    - async_ret.py
    - aiocrawl.py