章节导学

实现支持异步任务的线程池

使用Python3

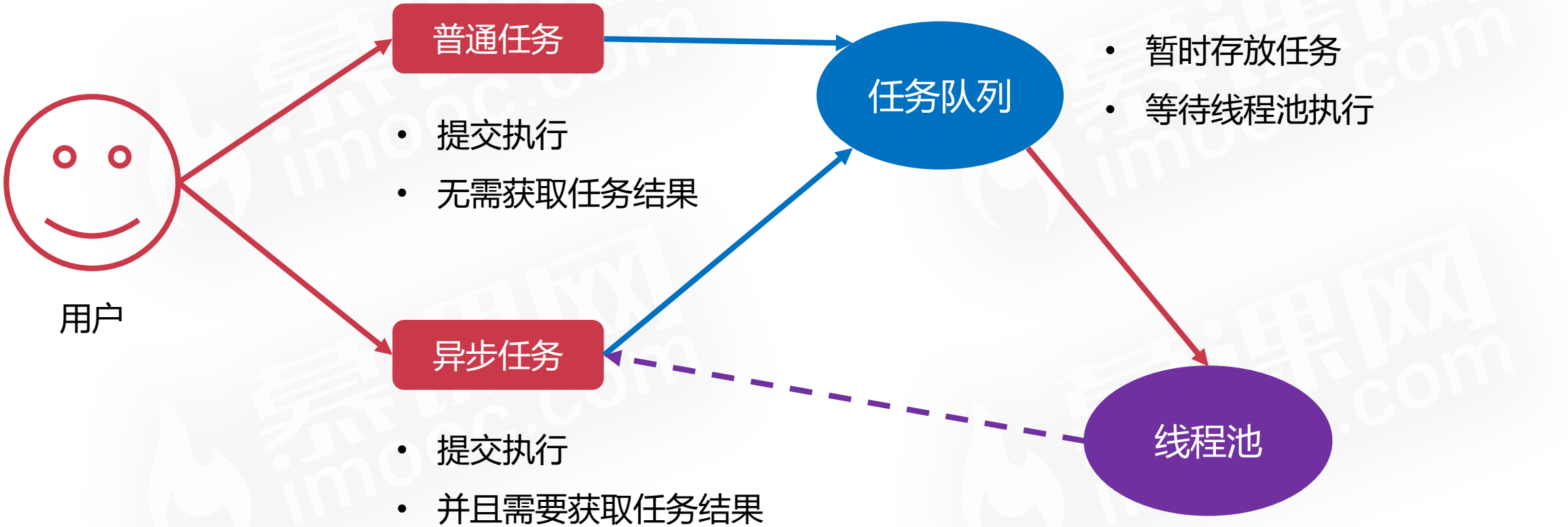
理解原理，自行实现

章节导学

- Java: ThreadPoolExecutor
- Python3: ThreadPoolExecutor
- 很多语言都提供了线程池

不再应该再局限于如何使用

章节导学


普通任务
提交执行
无需获取任务结果
任务队列
暂时存放任务
等待线程池执行
用户
异步任务
提交执行
并且需要获取任务结果
线程池
不断从任务队列取任务执行

章节导学


了解Python的同步原语
实现线程安全的队列
实现基本任务对象
了解线程池
实现任务处理线程
实现任务处理线程池
实现异步任务处理对象

Python同步原语

- 互斥锁
- 条件变量

Python同步原语

lock = threading.Lock()

lock.acquire()

lock.release()

互斥锁

Python同步原语

condition = threading.Condition()

condition.acquire()

condition.wait()

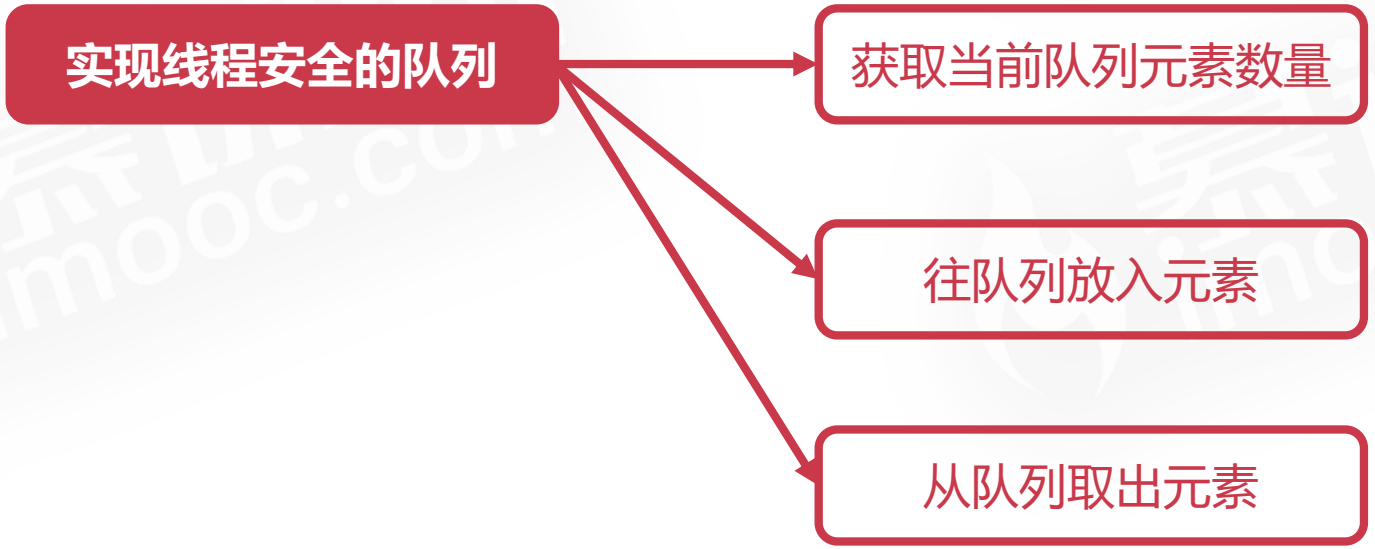
condition.release()

condition.notify()

条件变量

实现线程安全的队列Queue

- 队列用于存放多个元素，是存放各种元素的“池”



实现线程安全的队列Queue

- 队列可能有多个线程同时操作，因此需要保证线程安全

实现线程安全的队列Queue

多个线程同时访问队列元素

保证多个线程获取的串行

使用“锁”保护队列

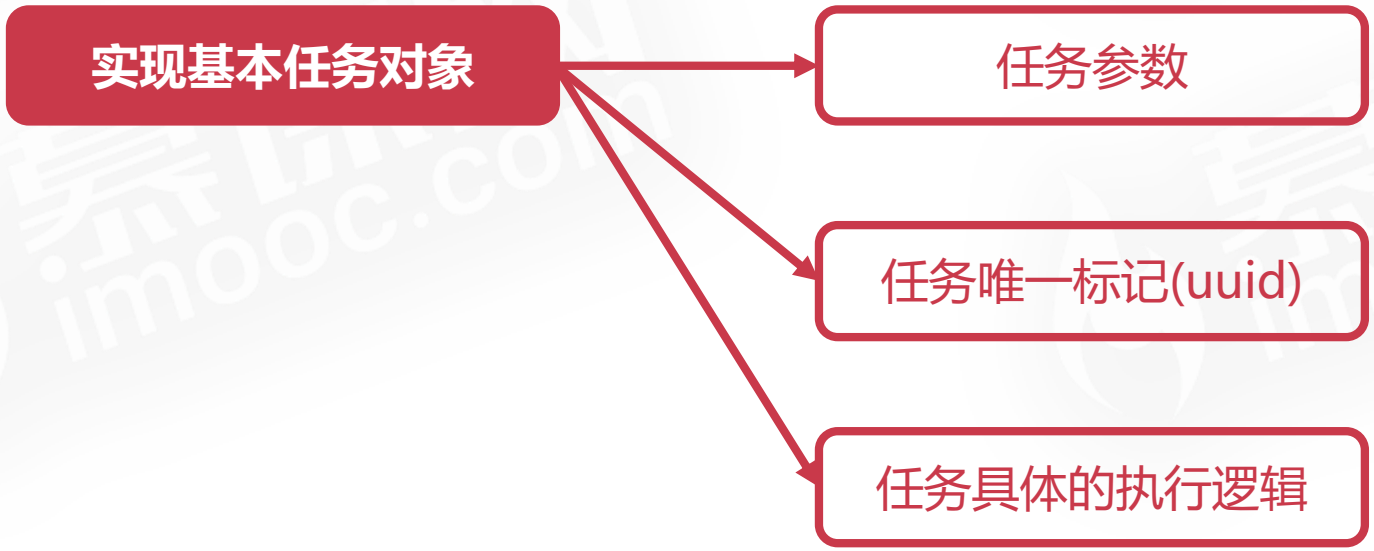
队列元素为空时获取队列元素

阻塞，等待队列不为空

使用条件变量等待队列元素

实现基本任务对象Task

- 任务处理逻辑



线程池简介

- 什么是线程池
- 为什么使用线程池

线程池简介

什么是线程池

- 线程池是存放多个线程的容器
- CPU调度线程执行后不会销毁线程
- 将线程放回线程池重复利用

线程池简介

- 什么是线程池
- 为什么使用线程池

线程池简介

为什么使用线程池

- 线程是稀缺资源，不应该频繁创建和销毁
- 架构解耦，线程创建和业务处理解耦，更加优雅
- 线程池是使用线程的最佳实践

线程池简介

为什么使用线程池

3．【强制】线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。

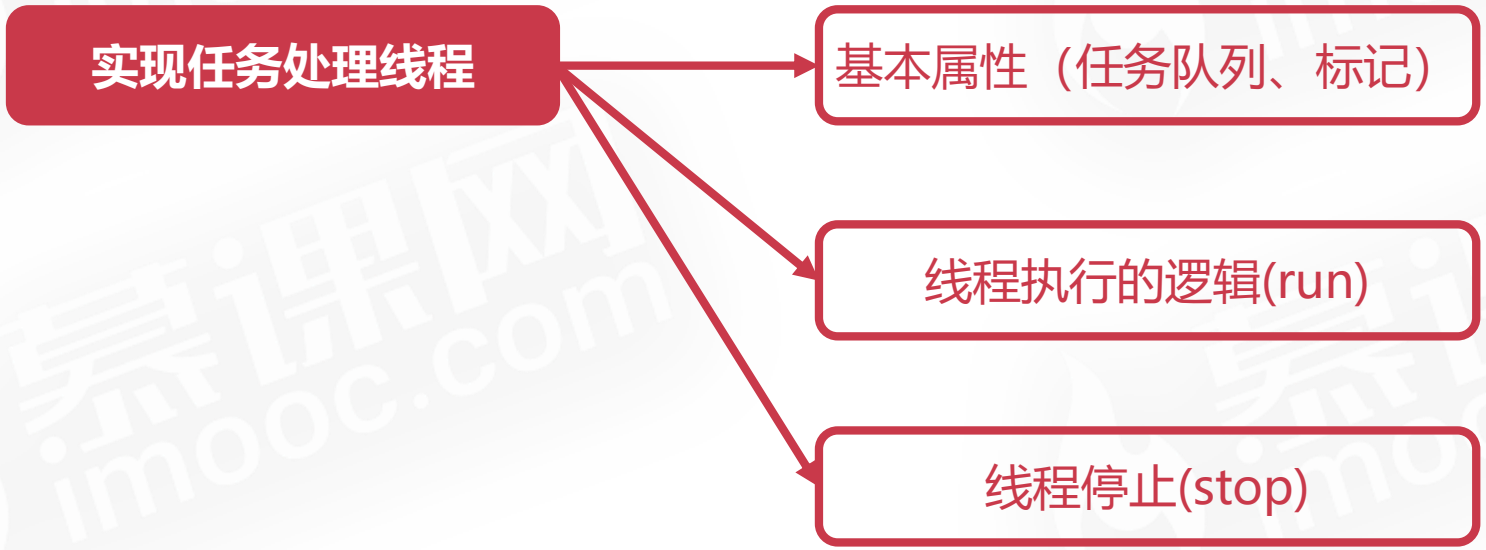
说明：使用线程池的好处是减少在创建和销毁线程上所花的时间以及系统资源的开销，解决资源不足的问题。如果不使用线程池，有可能造成系统创建大量同类线程而导致消耗完内存或者“过度切换”的问题。

阿里的Java手册就强制要求使用线程池

实现任务处理线程ProcessThread

- 任务处理线程需要不断的从任务队列里取任务执行
- 任务处理线程需要有一个标记，标记线程什么时候应该停止

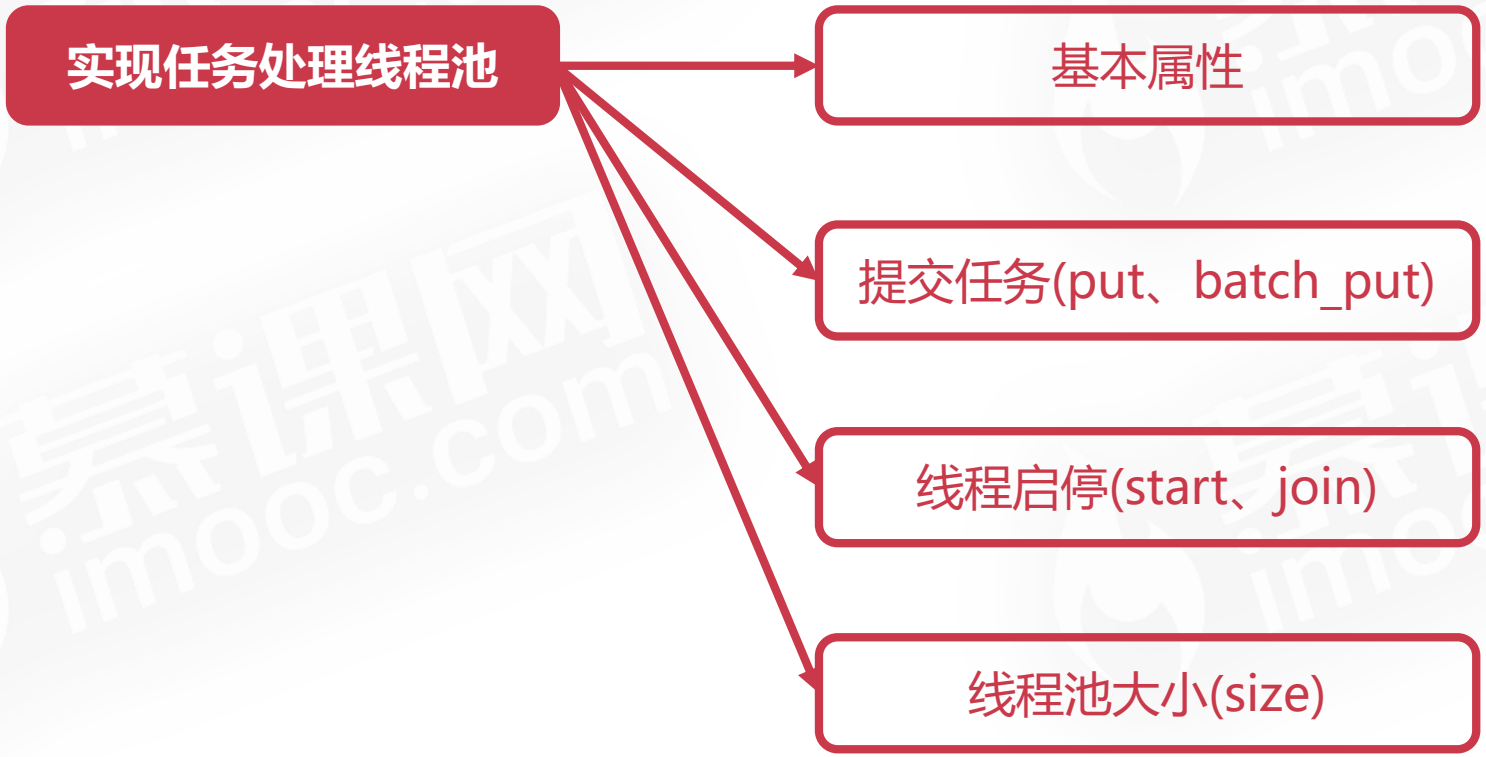
实现任务处理线程ProcessThread



实现任务处理线程池Pool

- 存放多个任务处理线程
- 负责多个线程的启停
- 管理向线程池的提交任务，下发给线程去执行

实现任务处理线程池Pool

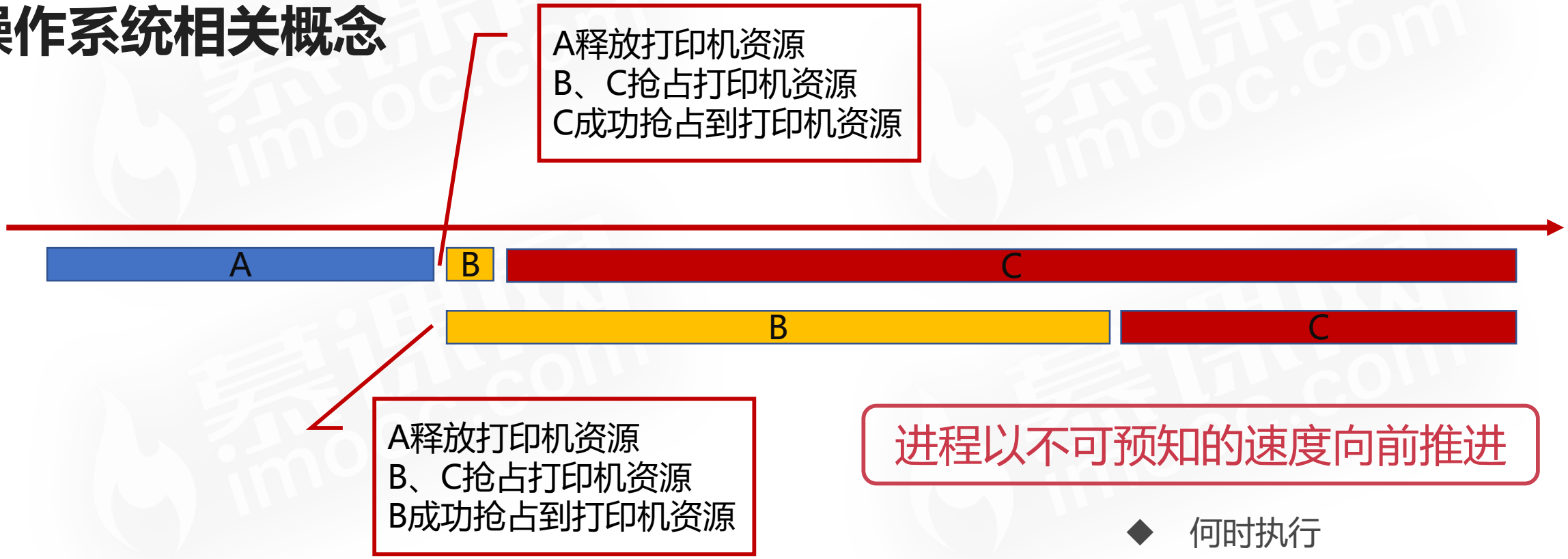


编写测试用例

实现异步任务处理AsyncTask

操作系统概览

操作系统相关概念



进程以不可预知的速度向前推进

- 何时执行
- 何时暂停
- 何时完成

异步性

实现异步任务处理AsyncTask

- 不知道任务什么时候执行
- 不知道任务什么时候执行完成

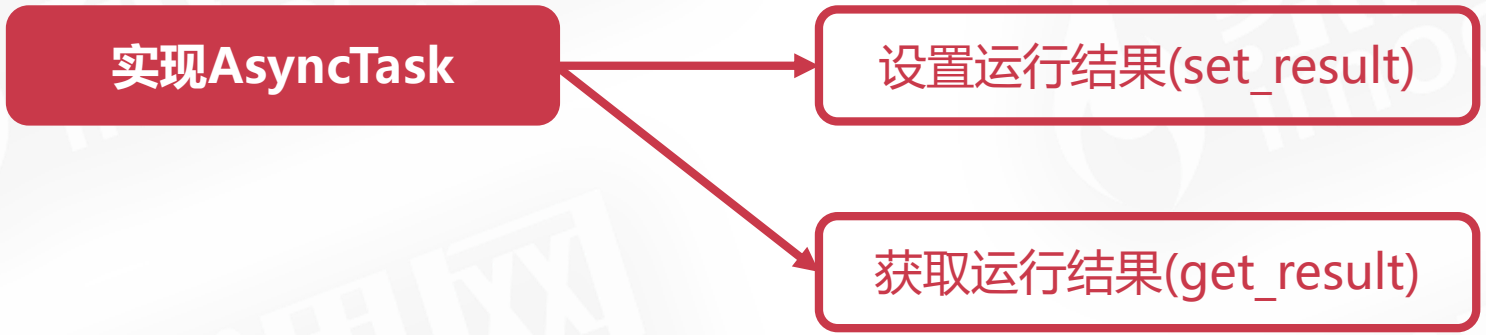
给任务添加一个标记，任务完成后，则标记为已完成

任务完成时，可直接获取任务运行结果

任务未完成时，获取任务结果，会阻塞获取线程

条件变量

实现异步任务处理AsyncTask



实现异步任务处理AsyncTask

Python GIL锁

实现异步任务处理AsyncTask

- 阅读各种语言所提供的异步功能的底层源码
- 对比其中的相同与不同