目标：掌握Golang协程调度原理，为什么GPG协程的调度更新快的

- 一、Golang“调度器”改造
  - 单进程时代的两个问题？
    - 1. 单一线程资源，计算密集只能一个任务一个任务处理
    - 2. 请相互重叠等待的IO任务顺序执行
  - 多计算密集型
    - 进程/线程的数量越多，切换成本就越大，收益就变差
  - 多进程，多线程的问题？
    - 多线程 隐 用户态协程（M:N，用多核原来的调度）
    - 多进程 共享内存
    - 内存占用小
    - 进程间通信
    - 进程占用内存
    - 进程/线程内存
    - 共享内存
    - 跨CPU调度资源
  - 协程(co-routine)，它是如何解？
    - M:1
    - 无法利用多个CPU
    - 当阻塞调用阻塞
    - 1:1
    - 最多线程 充分利用多核资源
    - 切换线程成本代价高昂
    - 最增利用多核
    - M:N
    - 这个最新协程调度器的优化实现
    - 改
  - Goroutine的优点
    - 内存占用
    - 几KB
    - 大量开辟
    - 启动速度
    - 切换成本低
  - 调度器的设计
    - 最多线程 充分利用多核资源
    - 早期的GPG调度器
    - 1. 单一锁，锁住一切
    - 2. 单线程 M:1 协程
    - 3. 多线程 M:N 协程
    - 4. 1:1 协程
    - 5. 多线程 M:N 协程
    - 6. 1:1 协程
    - 7. 多线程 M:N 协程
    - 8. 1:1 协程
    - 9. 多线程 M:N 协程
    - 1. 创建、销毁、调度协程需要整个M原锁住，这就形成了整的时间浪费。
    - 2. M跨核问题-各自线程间数据有时是相关的。
    - 3. 系统调用CPU时M之间的切换与协程管理的线程阻塞和恢复操作增加了系统开销。
    - 现在
    - Goroutine
    - 协程
    - Processor
    - 处理器
    - M thread
    - 内核线程
    - (1) GMP模型的基本
      - 全局队列
      - 存放等待运行的G
      - 存放等待运行的G
      - P的本地队列
      - 优先调度的G，最多256个
      - 数量限制
      - 不超过256G
      - 优先调度创建的G的P的本地队列，
      - 同步，如果满了会放全局队列中。
      - P的数
      - 程序启动时创建
      - 最多为GOMAXPROCS个
      - M的数
      - 当前操作系统分配给的CPU的线程数
      - P和M的关系
      - P代表逻辑CPU
      - 绑定M的CPU核心
      - 一个M只能绑定一个P
      - 一个P可以绑定多个M
      - 一个M最多只能绑定一个P
      - 一个P最多可以绑定多个M
      - 一个M最多可以绑定多个P
      - 一个M最多可以绑定多个P
      - 一个P最多可以绑定多个M