- 多线程
  - 任务和队列
    - 任务
      - 在线程执行的代码
      - 同步、异步
    - 队列
      - 队列指执行任务的等待队列，即用来存放任务的队列。队列是一种特殊的线性表，采用 FIFO（先进先出
      - 串行：每次只有一个任务执行，只开一个线程
      - 并发：多个任务同时执行，可开启多个线程
  - 进程线程
    - 进程是指在系统中正在运行的一个应用程序，系统进行资源分配和调度的基本单位，是操作系统结构的基础，每一个进程都有自己独立的虚拟内存空间。
    - 是程序执行流的最小单元线程是程序中一个单一的顺序控制流程。是进程内一个相对独立的、可调度的执行单元，是系统独立调度和分派CPU的基本单位指运行中的程序的调度单位
  - GCD
    - 内部原理
      - 优点
        - 多核的并行运算
        - 自动管理线程的生命周期（创建线程、调度任务、销毁线程
        - 不需要编写任何线程管理代码
      - 队列创建：dispatch_queue_create
        - 并发：DISPATCH_QUEUE_CONCURRENT
        - 串行：DISPATCH_QUEUE_SERIAL
      - 任务创建：
        - 同步：dispatch_sync
        - 异步：dispatch_async
      - 线程间通信
      - API
        - 栅栏方法：dispatch_barrier_async　　在执行完栅栏前面的操作之后，才执行栅栏操作，最后再执行栅栏后边的操作
        - 延时执行：dispatch_after
        - 快速迭代：dispatch_apply
          - 照指定的次数将指定的任务追加到指定的队列中，并等待全部队列执行结束。
          - 并发队列中执行的话顺序不一定
        - 队列组：dispatch_group
          - 当所有任务都执行完成之后，才执行dispatch_group_notify block 中的任务
          - dispatch_group_wait：当所有任务执行完成之后，才执行 dispatch_group_wait 之后的操作。但是，使用dispatch_group_wait 会阻塞当前线程
        - 信号量：dispatch_semaphore
          - 保持线程同步，将异步执行任务转换为同步执行任务
          - 保证线程安全，为线程加锁
      - 概念和宏定义
        - TSD(Thread-Specific Data) 表示线程私有数据
          - 作用：能够在同一个线程的不同函数中被访问
          - 实现：
            - pthread_key_create
            - pthread_setspecific
            - pthread_getspecific
        - fastpath && slowpath
          - fastpath(x) 依然返回 x，只是告诉编译器 x 的值一般不为 0，从而编译器可以进行优化。同理，slowpath(x) 表示 x 的值很可能为 0，希望编译器进行优化
          - __builtin_expect
        - DISPATCH_DECL
          - #define DISPATCH_DECL(name) typedef struct name##_s *name##_t
          - 快速定义一个指针，后缀_s, 指向同名后缀_t结构体
      - 数据结构
        - dispatch_object_t

```
1   typedef union {
2       struct dispatch_object_s *_do;          // dispatch_object_s结构体，这个是 GCD 的基类
3       struct dispatch_continuation_s *_dc;    // 任务类型，通常 dispatch_async内的block最终都会封装成这个数据类型
4       struct dispatch_queue_s *_dq;           // 任务队列，我们创建的对列都是这个类型的，不管是串行队列还是并发队列
5       struct dispatch_queue_attr_s *_dqa;     // 任务队列的属性，任务队列的属性里面包含了任务队列里面的一些操作函数，
可以表明这个任务队列是串行还是并发队列
6       struct dispatch_group_s *_dg;           // GCD的group
7       struct dispatch_source_s *_ds;          // GCD的sourece ，可以监测内核事件，文件读写事件和 socket 通信事件等
8       struct dispatch_source_attr_s *_dsa;    // sourece的属性。
9       struct dispatch_semaphore_s *_dsema;    // 信号量，如果了解过 pthread 都知道，信号量可以用来调度线程
10  } dispatch_object_t __attribute__((transparent_union));
```

          - _attribute__((transparent_union))是一种典型的 透明联合体，透明联合类型削弱了C语言的类型检测机制。
        - dispatch_object_s

```
struct dispatch_object_s {
    DISPATCH_STRUCT_HEADER(dispatch_object_s, dispatch_object_vtable_s);
};
```

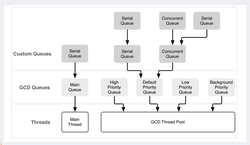
        - dispatch_continuation_s　　用于封装block和function
        - dispatch_once
          - typedef long dispatch_once_t;　长整型
          - dispatch_once_f
            - 如果已经完成，即DLOCK_ONCE_DONE，直接return
            - 首次进入dispatch_once_f，*val==0时候：
              - *val赋值为1，执行Block
              - 调用dispatch_atomic_barrier　　类似栅栏，保证编译器先执行Block在执行赋值
              - *val赋值为~0l;（按位补码，即全部都是1）
            - val不为1或者原子判断返回false，执行等待直到*val值为！0
          - 死锁情况
            - once的block中，使用还未完全初始化的单例对象，造成相互等待
            - point
              - 1、once初始化尽量简单
              - 2、减少相互依赖
              - 2、不要随意使用单例，避免内存占用
          - 问题：多个线程同时访问once时，利用什么机制做了容错?　　调用_dispatch_thread_event_wait阻塞线程，内部使用for(;;)自旋锁
        - dispatch_queue_s
          - dq_label　　队列名
          - DISPATCH_STRUCT_HEADER
            - 很多结构体中都用到这个宏
            - do_invoke，唤醒队列
            - do_dispose，销毁队列
            - do_probe，用户自定义的队列该字段为空，但是全局队列是
            - _dispatch_queue_wakeup_global

```
#define DISPATCH_STRUCT_HEADER(x, y)        \
6       const struct y *do_vtable;      \   // 这个结构体内包含了这个 dispatch_object_s 的操作函数
7       struct x *volatile do_next;     \   // 链表的 next
8       unsigned int do_ref_cnt;        \   // 引用计数
9       unsigned int do_xref_cnt;       \   // 外部引用计数
10      unsigned int do_suspend_cnt;        \   // suspend计数，用作暂停标志，比如延时处理的任务，设置
该引用计数之后；在任务到时后，计时器处理将会将该标志位修改，然后唤醒队列调度
11      struct dispatch_queue_s *do_targetq;\   // 目标队列，就是当前这个struct x在哪个队列运行
12      void *do_ctxt;                      \   // 上下文，我们要传递的参数，封装的任务block
13      void *do_finalizer
```

          - DISPATCH_QUEUE_HEADER
            - 队列头
            - dq_running: 队列正在运行的任务数。
            - dq_width: 队列的宽度(串行队列为1，并发队列应该大于1)。
            - dq_item_tail: 指向这个队列的尾节点。
            - dq_item_head: 指向这个队列的头节点。
            - dq_serialnum:
            - dq_finalizer_ctxt: 结束函数的的上下文(参数)。
            - dq_finalizer_func: 结束函数。
            - dq_serialnum
              - 操作_dispatch_queue_serial_numbers，并赋值给dq_serialnum，每个队列都是唯一的

```
// skip zero
// 1 – main_q
// 2 – mgr_q
// 3 – _unused_
// 4,5,6,7,8,9 – global queues
// we use 'xadd' on Intel, so the initial value == next assigned
```

        - dispatch_queue
          - dispatch_get_main_queue
            - 宏，返回结构体_dispatch_main_q的指针
            - _dispatch_main_q　　do_ref_cnt和do_xref_cnt值均为DISPATCH_OBJECT_GLOBAL_REFCNT，在retain和release操作中均直接返回，意味着生命周期伴随应用
          - _dispatch_mgr_q
            - 管理队列，GCD内部使用，不对外公开
            - 引用计数参考主队列
          - dispatch_get_global_queue
            - do_ctxt，上下文　　_dispatch_root_queue_contexts　　dgq_thread_mediator指向_dispatch_thread_mediator
            - 定义6个全局队列
            - .do_vtable全部指向&_dispatch_queue_root_vtable,
            - _dispatch_thread_mediator
          - dispatch_queue_create
            - 自定义队列,根据参数从全局队中获取_dispatch_root_queues，并赋值给target queue，有不同的优先级
            - 自定义队列的do_probe均为空，链接在rootqueue下，使用全局队列调度任务
            - 初始化_dispatch_queue_init
              - do_targetq = _dispatch_get_root_queue



              - 内联函数：类似宏，编译阶段代码替换，短小代码效率增加，长逻辑效率低下。
            - dispatch_queue_attr_t
              - 特殊处理
              - overcommit　　每次提交任务都开一个新线程，不会造成单个线程过载
            - 定义block，对block进行retain操作
        - dispatch_async
          - 主要思想：使用链表存储任务(block),并在线程池中取出任务一一执行。
          - 1、调用_dispatch_Block_copy，进行block的copy，并返回ctxt
          - 2、调用：dispatch_async_f，入参dq，ctxt，和_dispatch_call_block_and_release的函数指针
            - 1、获取dispatch_continuation_t dc　　调用_dispatch_continuation_alloc_cacheonly
            - 2、获取不到，调用_dispatch_async_f_slow
            - 3、使用dc封装ctxt和func，do_vtable为DISPATCH_OBJ_ASYNC_BIT
            - 3.5、如果有目标队列(do_targetq), 执行_dispatch_async_f2转发
            - 4、调用_dispatch_queue_push，把封装好的dc压入队列(dc可以是任务，也可以是队列)
              - 队列不为空，直接插入队尾，并重定向队尾指针
              - 队列为空
                - _dispatch_retain
                - _dispatch_wakeup
                - 主队列:_dispatch_queue_wakeup_main　　_dispatch_send_wakeup_main_thread，未开源
                - 全局队列:_dispatch_queue_wakeup_global
                - 1、全局队列检测、初始化、配置
                - 2、发送信号量,线程保活
                - 3、检测线程池大小、pthread_create创建线程。　　_dispatch_worker_thread
                - 线程调度_dispatch_worker_thread2　　原理_dispatch_worker_thread2
                - 取出队列中的一个内容_dispatch_queue_concurrent_drain_one
                - 1、队列只有一个内容，返回
                - 2、队列有多个内容，_dispatch_queue_wakeup_global　　创建一个新的线程，继续执行任务。
                - 对取出的任务进行处理_dispatch_continuation_pop
                - 1、入参是队列，调用_dispatch_queue_invoke处理队列　　1、队列空闲的话就唤醒
                - 2、是任务，直接执行。dc->dc_func(dc->dc_ctxt);
                - 线程调度结束之后，信号量等待。类似runloop，避免频繁调度消耗cpu
                - 等待65秒，等待结束后没有任务，线程退出并销毁
                - 自定义队列
                - 调用_dispatch_queue_push，将自定义队列压进它的targetq
                - 自定义队列的targetq是_dispatch_get_root_queue，即全局队列
                - 即自定义队列的任务最终会去到全局队列的分支里
                - _dispatch_release
        - dispatch_sync
          - 主队列执行_dispatch_sync_slow　　dispatch_sync_f
          - 否则执行：dispatch_sync_f
            入参函数：Block_invoke
            - 1、串行队列，执行dispatch_barrier_sync_f
              - 1、上锁失败(一个串行队列只允许lock一次)　　_dispatch_sync_f_slow　　_dispatch_sync_wait　　1、检测死锁：同一个串行队列被同一个线程做两次lock
              - 2、挨个执行任务，执行时使用信号量做互斥
            - 2、并发队列
              - 1、队列挂起状态或者有其他任务（没轮到你执行），_dispatch_sync_f_slow
              - 2、队列为空
                - 1、没有正在执行的任务　　唤醒队列_dispatch_wakeup
                - 2、有正在执行的任务（没轮到你执行）　　_dispatch_sync_f_slow
                - 将信号量压进它的形参队列中，然后等待这个信号量
                - 并发队列执行较快，因此很快就直接执行任务；
                - 便用
              - 3、执行任务func(ctxt);
        - dispatch_semaphore
          - 结构体dispatch_semaphore_t　　dsema_value：存储初始化时传的值，信号量做锁使用时传1
          - dispatch_semaphore_create　　dsema_value和dsema_orig，都赋值为value
          - dispatch_semaphore_wait　　1、信号量减一
            - value大于0，立即返回
            - 否则调用系统方法：semaphore_wait，直到收到signal调用
          - dispatch_semaphore_signal　　信号量 +1
            - 如果value大于0，说明没有线程在阻塞，直接return
            - semaphore_signal唤醒线程
          - _dispatch_semaphore_dispose
            - 信号量释放
            - 在信号量还在使用的时候、即阻塞某个线程的时候，将其置为空，会导致崩溃
          - 内核信号量原理
        - dispatch_group
          - dispatch_group_s
            - 本质是信号量
            - dg_waiters:
            - dg_notify_head
            - dg_notify_head
          - dispatch_group_create　　_dispatch_group_create_with_count　　与创建信号量基本一致
          - dispatch_semaphore_wait　　与信号量基本一致
      - 内部原理
      - API原理