线程

1、线程池的好处？ 四种线程池的使用场景，线程池的几个参数的理解？

- 参考回答：
    - 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则“过度切换”的问题，归纳总结就是
        - **重用存在的线程，减少对象创建、消亡的开销，性能佳。**
        - **可有效控制最大并发线程数，提高系统资源的使用率，同时避免过多资源竞争，避免堵塞。**
        - **提供定时执行、定期执行、单线程、并发数控制等功能。**
    - Android 中的线程池都是直接或间接通过配置 ThreadPoolExecutor 来实现不同特性的线程池.Android 中最常见的类具有不同特性的线程池分别为：
        - **newCachedThreadPool**：只有非核心线程，最大线程数非常大，所有线程都活动时会为新任务创建新线程,否

则会利用空闲线程（60s 空闲时间,过了就会被回收,所以线程池中有 0 个线程的可能)来处理任务.

        - 优点：任何任务都会被立即执行(任务队列 SynchronousQuue 相当于一个空集合);比较适合执行大量的耗时较少的任务.

    - **newFixedThreadPool**：只有核心线程，并且数量固定的，所有线程都活动时，因为队列没有限制大小，新任务会等待执行，当线程池空闲时不会释放工作线程，还会占用一定的系统资源。

        - 优点：更快的响应外界请求

    - **newScheduledThreadPool**：核心线程数固定,非核心线程（闲着没活干会被立即回收数）没有限制.

        - 优点：执行定时任务以及有固定周期的重复任务

    - **newSingleThreadExecutor**：只有一个核心线程,确保所有的任务都在同一线程中按序完成

        - 优点：不需要处理线程同步的问题

- 通过源码可以了解到上面的四种线程池实际上还是利用 ThreadPoolExecutor 类实现的

2、Android 中还了解哪些方便线程切换的类？

- 参考回答：
  - **AsyncTask**：底层封装了线程池和 Handler，便于执行后台任务以及在子线程中进行 UI 操作。
  - **HandlerThread**：一种具有消息循环的线程，其内部可使用 Handler。
  - **IntentService**：是一种异步、会自动停止的服务，内部采用 HandlerThread。

3、讲讲 AsyncTask 的原理

- 参考回答：
  - **AsyncTask 中有两个线程池（SerialExecutor 和 THREAD_POOL_EXECUTOR）和一个 Handler**

（InternalHandler），其中线程池 SerialExecutor 用于任务的排队，而线程池 THREAD_POOL_EXECUTOR 用于真正地执行任务，InternalHandler 用于将执行环境从线程池切换到主线程。

- **sHandler 是一个静态的 Handler 对象，为了能够将执行环境切换到主线程，这就要求 sHandler 这个对象必须在主线程创建。由于静态成员会在加载类的时候进行初始化，因此这就变相要求 AsyncTask 的类必须在主线程中加载，否则同一个进程中的 AsyncTask 都将无法正常工作。**

4、IntentService 有什么用 ？

- 参考回答：
  - **IntentService** 可用于**执行后台耗时**的任务，当任务执行完成后会**自动停止**，同时由于 IntentService 是服务的原因，不同于普通 Service，IntentService 可**自动创建子线程**来执行任务，这导致它的优先级比单纯的线程要高，不容易被系统杀死，所以 IntentService 比较适合执行一些**高优先级**的后台任务。

5、直接在 Activity 中创建一个 thread 跟在 service 中创建一个 thread 之间的区别？

- 参考回答：

- **在 Activity 中被创建**：该 Thread 的就是为这个 Activity 服务的，完成这个特定的 Activity 交代的任务，主动通知该 Activity 一些消息和事件，Activity 销毁后，该 Thread 也没有存活的意义了。
- **在 Service 中被创建**：这是保证最长生命周期的 Thread 的唯一方式，只要整个 Service 不退出，Thread 就可以一直在后台执行，一般在 Service 的 onCreate()中创建，在 onDestroy()中销毁。所以，在 Service 中创建的 Thread，适合长期执行一些独立于 APP 的后台任务，比较常见的就是：在 Service 中保持与服务器端的长连接。

6、ThreadPoolExecutor 的工作策略 ？

- 参考回答：ThreadPoolExecutor 执行任务时会遵循如下规则
  - 如果线程池中的线程数量**未达到**核心线程的数量，那么会直接启动一个核心线程来执行任务。
  - 如果线程池中的线程数量**已经达到**或则超过核心线程的数量，那么任务会被插入任务队列中排队等待执行。
  - 如果在第 2 点无法将任务插入到任务队列中，这往往是由于任务队列已满，这个时候如果在线程数量**未达到线程池规定的最大值**，那么会立刻启动一个非核心线程来执行任务。
  - 如果第 3 点中线程数量**已经达到线程池规定的最大值**，那么就拒绝执行此任务，ThreadPoolExecutor 会调用

RejectedExecutionHandler 的 rejectedExecution 方法来通知调用者。

7、Handler、Thread 和 HandlerThread 的差别？

- 参考回答：
  - **Handler**：在 android 中负责发送和处理消息，通过它可以实现其他支线线程与主线程之间的消息通讯。
  - **Thread**：Java 进程中执行运算的最小单位，亦即执行处理机调度的基本单位。某一进程中一路单独运行的程序。
  - **HandlerThread**：一个继承自 Thread 的类 HandlerThread，Android 中没有对 Java 中的 Thread 进行任何封装，而是提供了一个继承自 Thread 的类 HandlerThread 类，这个类对 Java 的 Thread 做了很多便利的封装。HandlerThread 继承于 Thread，所以它本质就是个 Thread。与普通 Thread 的差别就在于，它在内部直接实现了 Looper 的实现，这是 Handler 消息机制必不可少的。有了自己的 looper，可以让我们在自己的线程中分发和处理消息。如果不用 HandlerThread 的话，需要手动去调用 Looper.prepare()和 Looper.loop()这些方法。

8、ThreadLocal 的原理

- 参考回答：

- ThreadLocal 是一个关于创建线程局部变量的类。使用场景如下所示：
    - **实现单个线程单例以及单个线程上下文信息存储，比如交易 id 等。**
    - **实现线程安全，非线程安全的对象使用 ThreadLocal 之后就会变得线程安全，因为每个线程都会有一个对应的实例。 承载一些线程相关的数据，避免在方法中来回传递参数。**
- 当需要使用多线程时，有个变量恰巧不需要共享，此时就不必使用 synchronized 这么麻烦的关键字来锁住，每个线程都相当于在堆内存中开辟一个空间，线程中带有对共享变量的缓冲区，通过缓冲区将堆内存中的共享变量进行读取和操作，ThreadLocal 相当于线程内的内存，一个局部变量。每次可以对线程自身的数据读取和操作，并不需要通过缓冲区与 主内存中的变量进行交互。并不会像 synchronized 那样修改主内存的数据，再将主内存的数据复制到线程内的工作内存。ThreadLocal 可以让线程独占资源，存储于线程内部，避免线程堵塞造成 CPU 吞吐下降。
- 在每个 Thread 中包含一个 ThreadLocalMap，ThreadLocalMap 的 key 是 ThreadLocal 的对象，value 是独享数据。

9、多线程是否一定会高效（优缺点）

- 参考回答：
  - 多线程的优点：
    - **方便高效的内存共享 - 多进程下内存共享比较不便，且会抵消掉多进程编程的好处**
    - **较轻的上下文切换开销 - 不用切换地址空间，不用更改 CR3 寄存器，不用清空 TLB**
    - **线程上的任务执行完后自动销毁**
  - 多线程的缺点：
    - **开启线程需要占用一定的内存空间(默认情况下,每一个线程都占 512KB)**
    - **如果开启大量的线程,会占用大量的内存空间,降低程序的性能**
    - **线程越多,cpu 在调用线程上的开销就越大**
    - **程序设计更加复杂,比如线程间的通信、多线程的数据共享**
  - 综上得出，多线程**不一定**能提高效率，在内存空间紧张的情况下反而是一种负担，因此在日常开发中，应尽量
    - **不要频繁创建，销毁线程，使用线程池**

- **减少线程间同步和通信（最为关键）**
- **避免需要频繁共享写的数据**
- **合理安排共享数据结构，避免伪共享（false sharing）**
- **使用非阻塞数据结构/算法**
- **避免可能产生可伸缩性问题的系统调用（比如 mmap）**
- **避免产生大量缺页异常，尽量使用 Huge Page**
- **可以的话使用用户态轻量级线程代替内核线程**

10、多线程中,让你做一个单例,你会怎么做

- 参考回答：
  - 多线程中建立单例模式考虑的因素有很多，比如线程安全 -延迟加载-代码安全:如防止序列化攻击，防止反射攻击(防止反射进行私有方法调用) -性能因素
  - 实现方法有多种，饿汉，懒汉(线程安全，线程非安全)，双重检查(DCL),内部类，以及枚举

```
//OkHttp例子
private static volatile OkHttpHelper sInstance;

public static OkHttpHelper getInstance() {
        if (sInstance == null) {
            synchronized (OkHttpHelper.class) {
                if (sInstance == null) {
                    sInstance = new OkHttpHelper();
                }
            }
        }
   return sInstance;
}
```

11、除了 notify 还有什么方式可以唤醒线程

- 参考回答：
    - 当一个拥有 Object 锁的线程调用 wait()方法时，就会使当前线程加入 object.wait 等待队列中，并且释放当前占用的 Object 锁，这样其他线程就有机会获取这个 Object 锁，获得 Object 锁的线程调用 notify()方法，就能在 Object.wait 等待队列中随机唤醒一个线程（该唤醒是随机的与加入的顺序无关，优先级高的被唤醒概率会高）
    - 如果调用 notifyAll（ ）方法就唤醒全部的线程。**注意:调用 notify()方法后并不会立即释放 object 锁，会等待该线程执行完毕后释放 Object 锁。**

12、什么是 ANR？什么情况会出现 ANR ？如何避免 ？ 在不看代码的情况下如何快速定位出现 ANR 问题所在 ？

- 参考回答：
    - ANR（Application Not Responding，应用无响应）：当操作在一段时间内系统无法处理时，会在系统层面会弹出 ANR 对话框
    - 产生 ANR 可能是因为 5s 内无响应用户输入事件、10s 内未结束 BroadcastReceiver、20s 内未结束 Service
    - 想要避免 ANR 就不要在主线程做耗时操作，而是通过开子线程，方法比如继承 Thread 或实现 Runnable 接口、使用 AsyncTask、IntentService、HandlerThread 等