目录

1. AsyncTask 如何使用?(源码)

AsyncTask：封装了线程池和 Handler，方便开发者在子线程中更新 UI

AsyncTask 使用方法：

使用中的三个参数 ：

1) Params（传入参数）：表示后台任务执行时的参数类型，该参数会传给 AysncTask 的 doInBackground()方法

2) Progress（进度）：表示后台任务的执行进度的参数类型，该参数会作为 onProgressUpdate()方法的参数

3) Result（结果）：表示后台任务的返回结果的参数类型，该参数会作为 onPostExecute()方法的参数

使用中的六个方法 ：

1) Execute（）：用来用来执行一个异步任务，就是实现的 AsyncTask 的类调用的。

2) onPreExecute()：异步任务开启之前回调，在主线程中执行。调用后立即执行

3) doInBackground()：执行异步任务，在线程池中执行。 在 onPreExecute 完成后立即执行，用于执行较为费时的操作，此方法

将接收输入参数和返回计算结果。方法中不可以更新 UI。

4) onProgressUpdate()：当 doInBackground 中调用 publishProgress 时回调，在主线程中执行。

5) onPostExecute():在异步任务执行之后回调,在主线程中执行。后台结束时候调用的方法，会返回结果。注意，不能执行多次，不然会报错，且必须在 UI 线程中调用。

6) onCancelled()：在异步任务被取消时回调

2. 谈谈多线程在 Android 中的使用(源码)

https://www.jianshu.com/p/2b634a7c49ec

Android 提供了四种常用的操作多线程的方式，分别是：

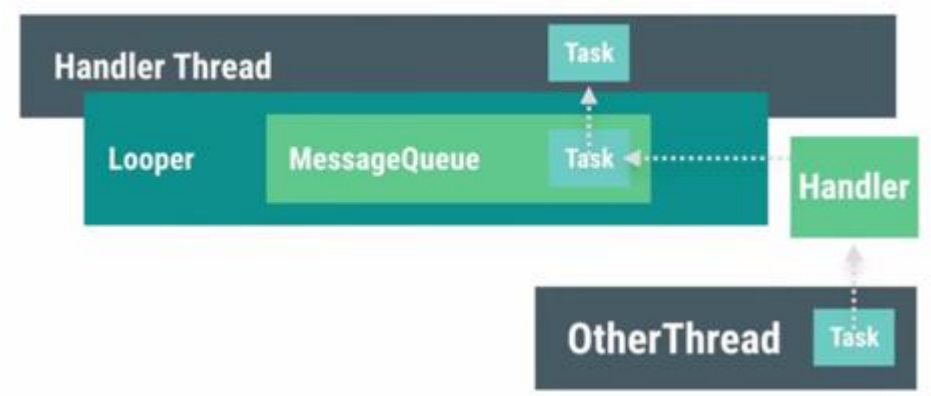
1. Handler+Thread
2. AsyncTask
3. ThreadPoolExecutor
4. IntentService

Handler + Thread

Android 主线程包含一个消息队列(MessageQueue)，该消息队列

里面可以存入一系列的 Message 或 Runnable 对象。通过一个 Handler 你可以往这个消息队列发送 Message 或者 Runnable 对象，并且处理这些对象。

每次你新创建一个 Handler 对象，它会绑定于创建它的线程(也就是 UI 线程)以及该线程的消息队列，从这时起，这个 handler 就会开始把 Message 或 Runnable 对象传递到消息队列中，并在它们出队列的时候执行它们。



Handler Thread 原理图

Handler（会绑定于创建它的线程(也就是 UI 线程)以及该线程的消息队列）可以把一个 Message 对象或者 Runnable 对象压入到消息队列中，进而在 UI 线程中获取 Message 或者执行 Runnable 对象。Handler 把压入消息队列有两类方式，Post 和 sendMessage：

优缺点

1. Handler 用法简单明了，可以将多个异步任务更新 UI 的代码放在一起，清晰明了

2. 处理单个异步任务代码略显多

适用范围

1. 多个异步任务的更新 UI

AsyncTask

AsyncTask 是 android 提供的轻量级的异步类,可以直接继承 AsyncTask，在类中实现异步操作，并提供接口反馈当前异步执行的程度(可以通过接口实现 UI 进度更新)，最后反馈执行的结果给 UI 主线程。内部通过 Handler + Thread 原理。

AsyncTask 通过一个阻塞队列 BlockingQuery<Runnable>存储待执行的任务，利用静态线程池 THREAD_POOL_EXECUTOR 提供一定数量的线程，默认 128 个。默认采用串行任务执行器，通过静态串行任务执行器 SERIAL_EXECUTOR 控制任务串行执行，循环取出任务交给 THREAD_POOL_EXECUTOR 中的线程执行，执行完一个，再执行下一个。

优缺点

1. 处理单个异步任务简单，可以获取到异步任务的进度
2. 可以通过 cancel 方法取消还没执行完的 AsyncTask
3. 处理多个异步任务代码显得较多

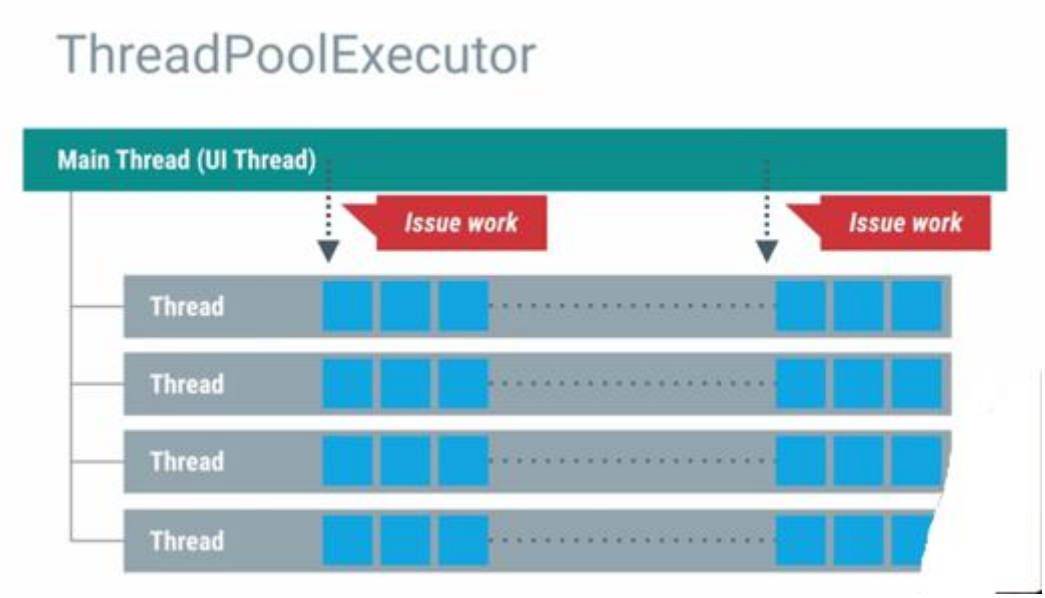
适用范围

单个异步任务的处理

ThreadPoolExecutor

ThreadPoolExecutor 提供了一组线程池，可以管理多个线程并

行执行。这样一方面减少了每个并行任务独自建立线程的开销，另一方面可以管理多个并发线程的公共资源，从而提高了多线程的效率。所以 ThreadPoolExecutor 比较适合一组任务的执行。Executors 利用工厂模式对 ThreadPoolExecutor 进行了封装，使用起来更加方便。



适用范围

1. 批处理任务

IntentService

IntentService 继承自 Service，是一个经过包装的轻量级的 Service，用来接收并处理通过 Intent 传递的异步请求。客户端通过调用 startService(Intent)启动一个 IntentService，利用一个 work 线程依次处理顺序过来的请求，处理完成后自动结束 Service。

特点

1. 一个可以处理异步任务的简单 Service

3. 进程和 Application 的生命周期（源码）

https://blog.csdn.net/timshinlee/article/details/73770821

Android 一个不同寻常的基本特点就是应用的进程生命周期并不是由应用本身控制的，而是由系统综合考虑决定的，因素包括系统所知的该应用运行中的组件、对于用户的重要程度，以及系统所有的可用内存大小。

Android 根据运行在进程中的组件及其状态来决定进程的优先级。可分为以下几种：

1. **前台进程：**即用户正在交互的进程。多种应用组件都可以使所在进程变成前台进程。（与用户正在交互的 activity、正在运行 BroadcasetReceiver 中的 onReceiver、一个 Service 正在执行回调）

前台进程在系统内只有少数几个，内存低的情况下是最后才会被终止掉的。

2. **可见进程：**即正在执行用户意识到的任务的进程，所以终止这个进程会导致明显的负面体验。（一个可见不可操作的 activity、正在运行的前台服务（通过 Service.startForeground()）、系统使用的服务（动态壁纸等、输入法服务等））

3. **服务进程：**含有一个调用了 Service.startService()方法启动的服务。尽管这种进程通常对于用户不可见，但是可能运行一些用户需要的工作。长期运行的服务（比如 30 分钟以上）会被降级到

缓存进程的级别。可以避免一些内存泄露或者其他问题的长期运行服务消耗系统资源。

4. **缓存进程：**是当前不需要的，系统可以随时终止的进程。在一个良好的系统里，总是有多个可用的缓存进程以便切换，只在需要时终止最久的进程。只有在非常严峻的情况下，所有的缓存进程都被终止了，必须开始终止服务进程了。 （不可见的 activity 实例）

对进程进行分类时，是基于进程内所有正在运行的组件的最高优先级来决定。一个进程的优先级也可能基于其他依赖的进程优先级而提升。

4. looper 架构(源码)

https://blog.csdn.net/hengqiaqia/article/details/78335239

我们的主线程（UI 线程）就是一个消息循环的线程。安卓引入一个新的机制 Handle，我们有消息循环，就要往消息循环里面发送相应的消息，消息的发送和清除、消息的的处理这些都封装在 Handle 里面，注意 Handle 只是针对那些有 Looper 的线程，不管是 UI 线程还是子线程，只要你有 Looper，我就可以往你的消息队列里面添加东西，并做相应的处理。

但是这里还有一点，就是只要是关于 UI 相关的东西，就不能放在子线程中，因为子线程是不能操作 UI 的，只能进行数据、系统等其他非 UI 的操作。

Looper.myLooper();获得当前的 Looper

Looper.getMainLooper () 获得 UI 线程的 Lopper

我们看看 Handle 的初始化函数，如果没有参数，那么他就默认使用的是当前的 Looper，如果有 Looper 参数，就是用对应的线程的 Looper。

如果一个线程中调用 Looper.prepare()，那么系统就会自动的为该线程建立一个消息队列，然后调用 Looper.loop();之后就进入了消息循环，这个之后就可以发消息、取消息、和处理消息。这个如何发送消息和如何处理消息可以再其他的线程中通过 Handle 来做，但前提是我们的 Hanle 知道这个子线程的 Looper，但是你如果不是在子线程运行 Looper.myLooper()，一般是得不到子线程的 looper 的。

5. SP（SharedPreferences）是进程同步的吗?有什么方法做到同步？(源码)

1. SharedPreferences 不支持进程同步

一个进程的情况，经常采用 SharePreference 来做，但是 SharePreference 不支持多进程，它基于单个文件的，默认是没有考虑同步互斥，而且，APP 对 SP 对象做了缓存，不好互斥同步.

MODE_MULTI_PROCESS 的作用是什么?

在 getSharedPreferences 的时候，会强制让 SP 进行一次读取操作，从而保证数据是最新的. 但是若频繁多进程进行读写 . 若某个进程持有了一个外部 sp 对象，那么不能保证数据是最新的. 因为刚刚被别的进程更新了.

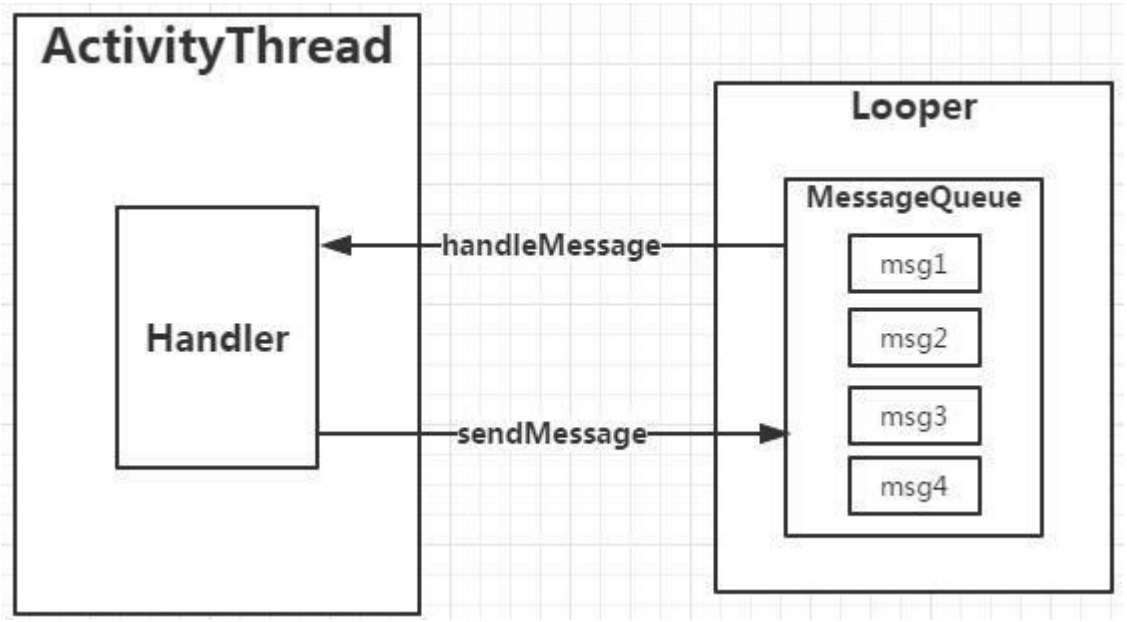
2.考虑用 ContentProvider 来实现 SharedPreferences 的进程同步.ContentProvider 基于 Binder，不存在进程间互斥问题，对于同步，也做了很好的封装，不需要开发者额外实现。

另外 ContentProvider 的每次操作都会重新 getSP. 保证了 sp 的一致性.

6. Handler 机制和底层实现（原理）

https://www.cnblogs.com/ryanleee/p/8204450.html

https://www.jianshu.com/p/b03d46809c4d



上面一共出现了几种类，ActivityThread，Handler，MessageQueue，Looper，msg（Message）,对这些类作简要介绍：

ActivityThread：程序的启动入口，该类就是我们说的主线程，它对 Looper 进行操作的。

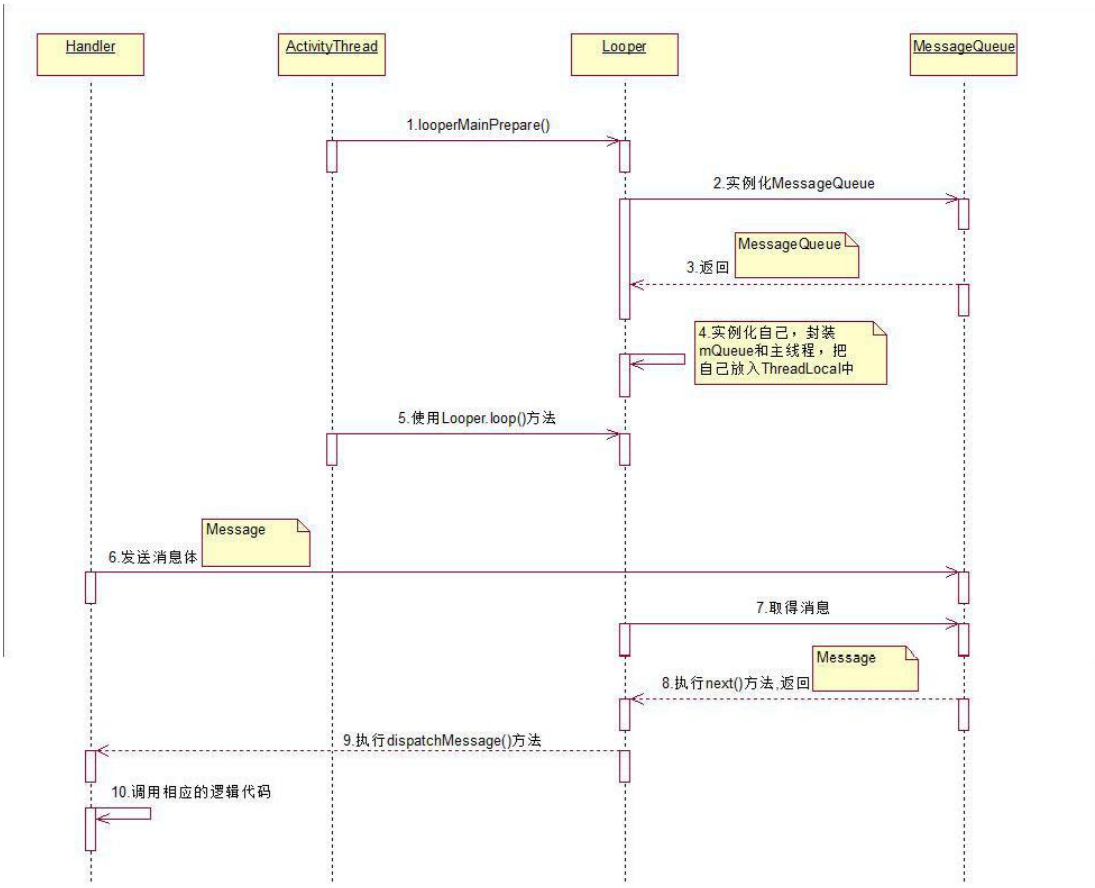
Handler：字面意思是操控者，该类有比较重要的地方，就是通过 handler 来发送消息（sendMessage）到 MessageQueue 和 操作控件的更新（handleMessage）。handler 下面持有这 MessageQueue 和 Looper 的对象。

MessageQueue：字面意思是消息队列，就是封装 Message 类。对 Message 进行插入和取出操作。

Message：这个类是封装消息体并被发送到 MessageQueue 中的，给类是通过链表实现的，其好处方便 MessageQueue 的插入和取出操作。还有一些字段是（int what，Object obj，int arg1，int arg2）。

what 是用户定义的消息和代码，以便接收者（handler）知道这个是关于什么的。obj 是用来传输任意对象的，arg1 和 arg2 是用来传递一些简单的整数类型的。

下面，我们按照启动顺序来进行源码分析：



先获取 looper，如果没有就创建

创建过程：

ActivityThread 执行 looperMainPrepare（），该方法先实例化 MessageQueue 对象，然后实例化 Looper 对象，封装 mQueue 和主线程，把自己放入 ThreadLocal 中

再执行 loop()方法，里面会重复死循环执行读取 MessageQueue。（接着 ActivityThread 执行 Looper 对象中的 loop（）方法）

此时调用 sendMessage（）方法，往 MessageQueue 中添加数据，其取出消息队列中的 handler，执行 dispatchMessage（），进而执行 handleMessage（），Message 的数据结构是基于链表的

7. Handler、Thread 和 HandlerThread 的差别（原理）

https://blog.csdn.net/lmj623565791/article/details/47079737/

Handler

Handler 是 Android 中引入的一种让开发者参与处理线程中消息循环的机制。每个 Hanlder 都关联了一个线程，每个线程内部都维护了一个消息队列 MessageQueue，这样 Handler 实际上也就关联了一个消息队列。

可以通过 Handler 将 Message 和 Runnable 对象发送到该 Handler 所关联线程的 MessageQueue（消息队列）中，然后该消息队列一直在循环拿出一个 Message，对其进行处理，处理完之后拿出下一个 Message，继续进行处理，周而复始。

当创建一个 Handler 的时候，该 Handler 就绑定了当前创建 Hanlder 的线程。从这时起，该 Hanlder 就可以发送 Message 和 Runnable 对象到该 Handler 对应的消息队列中，当从 MessageQueue 取出某个 Message 时，会让 Handler 对其进行处理。

作用：

Handler 可以用来在多线程间进行通信，在另一个线程中去更新 UI 线程中的 UI 控件只是 Handler 使用中的一种典型案例,除此之外，Handler 可以做很多其他的事情。Handler 是 Thread 的代言人，是多线程之间通信的桥梁，通过 Handler，我们可以在一个线程中控制另一个线程去做某事。

Thread

线程，可以看作是进程的一个实体，是 CPU 调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。

```
public class MyThread extends Thread{
@Override public void run(){
        super.run();
        // do something
    }
}
```

HandlerThread：封装了 Handler + Thread

内部原理 = Thread 类 + Handler 类机制，即：

通过继承 Thread 类，快速地创建 1 个带有 Looper 对象的新工作线程

通过封装 Handler 类，快速创建 Handler & 与其他线程进行通信。

内部调用 Looper.prepare();Looper.loop();在 Looper 对象的构造过程中，初始化了一个 MessageQueue，作为该 Looper 对象成员变量。

loop()就开启了，不断的循环从 MessageQueue 中取消息处理了，当没有消息的时候会阻塞，有消息的到来的时候会唤醒。

8. handler 发消息给子线程，looper 怎么启动？（原理）

什么问题呢。。发消息就是把消息塞进去消息队列，looper 在应用起来的时候已经就启动了，一直在轮询取消息队列的消息。

9. ThreadLocal 原理，实现及如何保证 Local 属性？（原理）

https://blog.csdn.net/singwhatiwanna/article/details/48350919

ThreadLocal：当某些数据是以线程为作用域并且不同线程具有不同的数据副本的时候，就可以考虑采用 ThreadLocal。（Looper、ActivityThread 以及 AMS 中都用到了）

如使用 ThreadLocal 可以解决不同线程不同 Looper 的需求。

虽然在不同线程中访问的是同一个 ThreadLocal 对象，但是它们通过 ThreadLocal 来获取到的值却是不一样的，这就是 ThreadLocal 的奇妙之处。ThreadLocal 之所以有这么奇妙的效果，是因为不同线程访问同一个 ThreadLocal 的 get 方法，ThreadLocal 内部会从各自的线程中取出一个数组，然后再从数组中根据当前 ThreadLocal 的索引去查找出对应的 value 值，很显然，不同线程中的数组是不同的，这就是为什么通过 ThreadLocal 可以在不同的线程中维护一套数据的副本并且彼此互不干扰。（从 ThreadLocal 的 set 和 get 方法可以看出，它们所操作的对象都是当前线程的 localValues 对象的 table 数组，因此在不同线程中访问同一个 ThreadLocal 的 set 和 get 方法，它们对 ThreadLocal 所做的读写操作仅限于各自线程的内部，这就是为什么 ThreadLocal 可以在多个线程中互不干扰地存储和修改数据。）

10. 请解释下在单线程模型中 Message、Handler、Message Queue、Looper 之间的关系（原理）

Handler 是 Android 官方给我们提供的一套更新 UI 线程的机制，也是一套消息处理机制，可以通过 Handler 来处理消息，更新 UI 等。

原理：

1. **Handler(消息的处理者和发送者)**：你可以构造 Handler 对象来与 Looper 沟通，以便 push 新消息到 Message Queue 里;或者接收 Looper 从 Message Queue 取出)所送来的消息。

2. **Message Queue(消息队列)**:用来存放线程放入的消息。读取会自动删除消息,单链表维护,在插入和删除上有优势。在其 next()中会无限循环，不断判断是否有消息，有就返回这条消息并移除。（Android 会自动替主线程(UI 线程)建立 Message Queue，但在子线程里并没有建立 Message Queue。所以调用 Looper.getMainLooper()得到的主线程的 Looper 不为 NULL，但调用 Looper.myLooper() 得到当前线程的 Looper 就有可能为 NULL）。

3. **Looper(轮循者)**：一个线程可以产生一个 Looper 对象，由它来管理此线程里的 Message Queue(消息队列),管理特定线程内对象之间的消息交换(Message Exchange)。（

1) 初始化循环检测：Looper 创建的时候会创建一个 MessageQueue，调用 loop()方法的时候消息循环开始，loop()也是一个死循环，会不断调用 messageQueue 的 next()，

2) 消息处理：当有消息就处理，那么就调用 handler 的 handlermessage 方法进行余下的操作。之所以这样做的原因是因为避免多线程并发更新 UI 线程所产生的问题的，如果我们允许其他子线程都可以更新界面，那么势必会造成界面

的错乱(因为没有加锁机制)，如果我们加锁，又会影响速度，所以，只能在主线程即 UI 线程里面更新界面。否则阻塞在 messageQueue 的 next()中。

3) 退出检测：当 Looper 的 quit() 被调用的时候会调 用 messageQueue 的 quit(),此时 next()会返回 null，然后 loop()方法也跟着退出。）

4. **ThreadLocal**：ThreadLocal 是线程内部的存储类，通过它可以实现在每个线程中存储自己的私有数据。即数据存储以后，只能在指定的线程中获取这个存储的对象，而其它线程则不能获取到当前线程存储的这个对象。负责存储和获取本线程的 Looper

5. **Message(消息的载体)**:一个 Bean 对象，里面的属性用来记录 Message 的各种信息。(Handler 处理完该 Message (update UI)后，Looper 则设置该 Message 为 NULL，以便回收！）

6. **主线程（UI 线程）**：当程序第一次启动时，Android 会同时启动一条主线程(Main Thread)。UI thread 通常就是 main thread，而 Android 启动程序时会替它建立一个 Message Queue。主要负责处理与 UI 相关的事件。

Handler(先进先出原则)通过调用 sendmessage 方法把消息放在消息队列 MessageQueue 中，Looper 负责把消息从消息队列中取出来，重新再交给 Handler 进行处理，三者形成一个循环。

https://blog.csdn.net/lovedren/article/details/51701477

11. 为什么不能在子线程更新 UI？（原理）

1. 子线程能在特定时间段更新 UI 的，只是不安全：

执行 onCreate 方法的那个时候 ViewRootImpl 对象还没创建，无法去检查当前线程。在访问 UI 的时候，ViewRoot 会去检查当前是哪个线程访问的 UI，如果不是主线程，就会抛出异常

2. 平时不能在子线程更新 UI：

目的在于提高移动端更新 UI 的效率和和安全性，以此带来流畅的体验。

Android 的 UI 访问是没有加锁的，多个线程可以同时访问更新操作同一个 UI 控件。也就是说访问 UI 的时候，android 系统当中的控件都不是线程安全的，这将导致在多线程模式下，当多个线程共同访问更新操作同一个 UI 控件时容易发生不可控的错误，而这是致命的。（UI 是非线程安全的，主线程和子线程同时更新 UI 的话会导致错误，如 UI 错乱之类的）所以 Android 中规定只能在 UI 线程中访问 UI，这相当于从另一个角度给 Android 的 UI 访问加上锁，一个伪锁。

12. Android 线程有没有上限？（原理）

其实这个没有上限的，因为资源都限制在这个进程里，你开多少线程都最多用这些资源。（暂无）

13. 线程池有没有上限？（原理）

http://www.trinea.cn/android/java-android-thread-pool/

https://blog.csdn.net/cfy137000/article/details/51422316

线程池的由来：

创建太多线程，将会浪费一定的资源，有些线程未被充分使用。销毁太多线程，将导致之后浪费时间再次创建它们。 创建线程太慢，将会导致长时间的等待，性能变差。 销毁线程太慢，导致其它线程资源饥饿

线程池从名字就可以看出，它是用来管理线程的，在线程池中，我们的线程不会被随意的创建出来，它可以缓存一定数量的线程，减少了资源的消耗，同时还可以指定线程的优先级，或者同时需要大量在耗时任务的时候，这些耗时操作是使用 FIFO(先进先出)还是 LIFO(后进先出)的策略。

总结来说，线程池的有点可以概括为以下三点：

重用线程池中的线程，避免因为线程的创建和销毁所带来的性能开销 能有效控制线程池的最大并发数，避免大量的线程之间因相互抢占系统资源而导致的阻塞现象。 能对线程进行简单的管理，并提供定时执行以及制定间隔循环执行等功能。安卓真正的线程池实现是

ThreadPoolExecutor，它提供了一系列参数来方便我们配置线程池，我们就来先看看这个 ThreadPoolExecutor 类线程池执行任务的规则

当向线程池中提交任务的时候，会满足以下规则：

1) 如果线程池中的线程数量没有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该任务。
2) 如果线程池中的线程数量已经达到核心线程数，那么任务就会被插入到任务队列中排队等待执行，当核心线程空闲的时候，就会从任务队列中按照某种规则取出一个任务来执行
3) 如果任务队列满了，或者由于其他原因，向线程池提交的任务不能插入到任务队列中的时候，这个时候就会去看线程池中的线程数是否达到线程池的上限，如果没有，就立即开启一个线程并执行。（增加非核心线程）
4) 如果线程池中正在工作的线程数已经达到了线程池设置的上限，此时再向线程池中提交任务，线程池就会拒绝执行此任务

四类线程池

FixedThreadPool

它是一种线程数量固定的线程池，它的核心线程和最大线程是相等的，即该线程池中的所有线程都是核心线程，所以它也并没有超时机制，而他的任务队列是无边界的任务队列，也就是可以添加无上限的任务，但是都会排队执行

CachedThreadPool

它的核心线程数是 0 也就是说该线程池并没有核心线程，该线程池的最大线程数是没有上限的（int 类型的上限），也就是说可以无限的创建线程。那么当新任务向线程池中提交的时候，如果有空闲线程，就会把任务放到空闲线程中去，如果没有空闲线程，就会开启一个新的线程来执行此任务，而它的队列 SynchronousQueue 是一个特殊的队列，在多数情况下，我们可以把它简单的理解为一个无法插入的队列（可能是直接创建线程执行了）。

比较适合执行大量的耗时少的任务，当线程池处于闲置状态的时候，线程池中的线程都会被销毁，这个时候该线程池几乎是不占用任何系统资源的

ScheduledThreadPool

可以看出它的核心线程数是固定的，而最大线程数没有限制（int 类型的上限），它与之前的线程池相区别的就是它的任务队列，DelayedWorkQueue 能让任务周期性的执行，也就是说该线程池可以周期性的执行任务。

SingleThreadExecutor

通过代码可以看出，这种线程池，就是 FixedThreadPool 但是实例化方法的参数是 1 的任务队列

通过观察官方为我们提供的几种线程池，我们发现，对于不同类型的线程池来说，决定他们有各种功能的最主要因素就是这个任务队列，而这个任务队列实际上是一个实现了叫 BlockingQueue 的对象，在这个接口里规定了加入或取出等方法，一共有 11 个方法，要复写起来非常麻烦，所幸，Java 也为我们封装了一些常用的实现类来方便我们的使用，常用的有以下几种

LinkedBlockingQueue：无界的队列

SynchronousQueue：直接提交的队列

DelayedWorkQueue：等待队列
PriorityBlockingQueue：优先级队列
ArrayBlockingQueue：有界的队列

LinkedBlockingQueue&ArrayBlockingQueue

这两个队列很像 LinkedList 和 ArrayList 就是一个是用数组实现的，一个使用链表实现的，它们都是 FIFO 的，而区别是 LinkedBlockingQueue 可以是没有数量上限的，而根据之间说的任务向现场池中添加的顺序我们知道，如果队列是无上限的话，线程池就不需要非核心线程了，可以看到 Java 封装好的线程池只要使用这个队列的，它的核心线程数和最大线程数都是一样的。

SynchronousQueue

这个队列会把任务直接提交给线程而不保持它们。在此，如果不存在可用于立即运行任务的线程，则试图把任务加入队列将失败，因此会构造一个新的线程。此策略可以避免在处理可能具有内部依赖性的请求集时出现锁。直接提交通常要求无界 maximumPoolSizes 以避免拒绝新提交的任务。当命令以超过队列所能处理的平均数连续到达时，简单说来，这种队列就是没什么用，走个过场而已，所以使用这个队列的时候，线程池的最大线程数一般是无上限的，也就是 int 类型的最大值

PriorityBlockingQueue

优先级队列，这种队列在向线程池中提交任务的时候会检测每一个任务的优先级，会先把优先级高的任务扔到线程池中，前几种队列我们都用过了，我们来写一个利用优先级队列的线程池。

在使用线程池的时候，我们确定线程池核心线程数的时候通常会根据 CPU 的核心数来确定的，通常会使用 CPU 核心数+1 来定为当前线程池的核心线程数，在 Android 中的 CPU 核心数可以通过 Runtime.getRuntime().availableProcessors();来获得

14. AsyncTask 机制（原理）

https://blog.csdn.net/yanbober/article/details/46117397

实质就是在一个线程池中执行，这个 THREAD_POOL_EXECUTOR 线程池是一个常量，也就是说整个 App 中不论有多少 AsyncTask 都只有这一个线程池。Handler + Thread 原理

15. AsyncTask 原理及不足（原理）

https://www.cnblogs.com/absfree/p/5357678.html

AsyncTask 是对 Handler 与线程池的封装。由于 AsyncTask 内部包含一个 Handler，所以可以发送消息给主线程让它更新 UI。另外，AsyncTask 内还包含了一个线程池。使用线程池的主要原因是避免不必要的创建及销毁线程的开销。

AsyncTask 的优点在于执行完后台任务后可以很方便的更新 UI，然而使用它存在着诸多的限制。先抛开内存泄漏问题，使用 AsyncTask 主要存在以下局限性：

1) 在 Android 4.1 版本之前，AsyncTask 类必须在主线程中加载，这意味着对 AsyncTask 类的第一次访问必须发生在主线程中；在 Android 4.1 以及以上版本则不存在这一限制，因为 ActivityThread（代表了主线程）的 main 方法中会自动加载 AsyncTask
2) AsyncTask 对象必须在主线程中创建
3) AsyncTask 对象的 execute 方法必须在主线程中调用
4) 一个 AsyncTask 对象只能调用一次 execute 方法

16. 如何取消 AsyncTask？（原理）

https://www.jianshu.com/p/0c6f4b6ed558

(1)**调用 cancel()：**但是他是在在 doInBackground（）之后执行

如果调用 cancel（）方法，它不会立即执行，只有当 doInBackground()方法执行完有返回值之后，会在 UI 主线程调用 cancel（），同时也会间接的调用 iscancelled（），并且返回 true ，这个时候就不会再调 onPostExecute（），然后在 doInBackground（）里定期检查 iscancelled（）方法的返回值，是否被 cancel，如果 return true，就尽快停止。

(2)**在耗时操作中设置一些 flag：**我们可以在这个线程中的耗时操作中设置一些 flag，也就是 AsyncTask 的 doInBackground 方法中的某些关键步骤。

然后在外层需要终止此线程的地方改变这个 flag 值，线程中的耗时代码一步步执行，当某一时刻发现 flag 的值变了，throwException，线程就不会再继续执行了。为了保险起见，在外层我们还要捕获这个异常，进行相应处理。（子线程被发生异常后会自己死掉而不会引起其他问题，更不会影响到主线程，更何况我们为了更加安全还捕获了异常并做处理）

我们找到了停止 Running 线程的办法，那么下次只要在 Activity

生命周期结束之前也结束掉线程的生命，就可以让你的应用程序更加安全健壮了。

17. Android 上的 Inter-Process-Communication 跨进程通信时如何工作的？

https://blog.csdn.net/SunshineTan/article/details/78649816

IPC 种类有四种：

1. **Binder 机制:**可以实现进程间通信。 Binder 类实现了 IBinder 接口。也可以理解为是一种虚拟的物理设备，设备驱动是 /dev/binder。

从 Android Framework 角度： Binder 是 ServiceManager 连接各种 Manager 和相应 ManagerService 的桥梁；

从 Android 应用层角度：Binder 是客户端和服务端进行通信的媒介。如 bindService，服务端会返回一个包含了服务端调用的 Binder 对象，客户端可以通过它获取服务端提供的服务或数据。

优点：

1) 高性能 ：从数据拷贝次数来看 Binder 只需要进行一次内存拷贝，而管道、消息队列、Socket 都需要两次，共享内存不需要拷贝，Binder 的性能仅次于共享内存。

2) 稳定性 ：Binder 基于 C/S 架构，客户端与服务端彼此独立，稳定性较好。

3) 安全性 ：Android 为每个应用分配了 UID，用来作为鉴别进程的重要标志，Android 内部也依赖这个 UID 进行权限管理，包括 6.0 以前的固定权限和 6.0 以后的动态权限，传送 IPC 只能由用户在数据包里填入UID/PID,这个标记完全 是在用户空间控制的，没有放在内核空间，因此有被恶意篡改的可能，因此 Binder 的安全性更高。

IBinder

IBinder 接口的实现类，该类用以提供客户端用来与服务进行交互的编程接口。

扩展 Binder 类

场景：服务仅供本地应用使用，不需要跨进程工作

原理：通过扩展 Binder 类并从 onBind() 返回它的一个实例来创建接口。客户端收到 Binder 后，可利用它直接访问 Binder 实现中以及 Service 中可用的公共方法。

实现：

1.创建 BindService 服务端，继承自 Service 并在类中，创建一个实现 IBinder 接口的实例对象并提供公共方法给客户端调用

2.从 onBind() 回调方法返回此 Binder 实例。

3.在客户端中，从 onServiceConnected() 回调方法接收 Binder，并使用提供的方法调用绑定服务。

2. Socket

可以实现任意两个终端/进程的通信。

3. Serializable、Parcelable

4. Linux 进程通信：

①管道：在创建时分配一个 page 大小的内存，缓存区大小比较有限；

②消息队列：信息复制两次，额外的 CPU 消耗；不合适频繁或信息量大的通信；

③共享内存：无须复制，共享缓冲区直接付附加到进程虚拟地址空间，速度快；但进程间的同步问题操作系统无法实现，必须各进程利用同步工具解决；

④套接字：作为更通用的接口，传输效率低，主要用于不通机器或跨网络的通信；

⑤**信号量：**常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。（

信号量（semaphore）的数据结构为一个值和一个指针，指针指向等待该信号量的下一个进程。信号量的值与相应资源的使用情况有关。

当它的值大于 0 时，表示当前可用资源的数量；

当它的值小于 0 时，其绝对值表示等待使用该资源的进程个数。

）

⑥**信号：**不适用于信息交换，更适用于进程中断控制，比如非法内存访问，杀死某个进程等；

跨进程通信

名　称	优　点	缺　点	适用场景
Bundle	简单易用	只能传输 Bundle 支持的数据类型	四大组件间的进程间通信
文件共享	简单易用	不适合高并发场景，并且无法做到进程间的即时通信	无并发访问情形，交换简单的数据实时性不高的场景
AIDL	功能强大，支持一对多并发通信，支持实时通信	使用稍复杂，需要处理好线程同步	一对多通信且有 RPC 需求
Messenger	功能一般，支持一对多串行通信，支持实时通信	不能很好处理高并发情形，不支持 RPC，数据通过 Message 进行传输，因此只能传输 Bundle 支持的数据类型	低并发的一对多即时通信，无 RPC 需求，或者无须要返回结果的 RPC 需求
ContentProvider	在数据源访问方面功能强大，支持一对多并发数据共享，可通过 Call 方法扩展其他操作	可以理解为受约束的 AIDL，主要提供数据源的 CRUD 操作	一对多的进程间的数据共享
Socket	功能强大，可以通过网络传输字节流，支持一对多并发实时通信	实现细节稍微有点烦琐，不支持直接的 RPC	网络数据交换

(1)　Intent 传递数据

在 Intent 中附加 extras 来传递信息。Activity、Service、Receiver 都支持 Intent 中传递 Bundle 数据。

Bundle 实现了 Parcelable 接口，可以在不同进程间传输。

Bundle 中的数据必须能够被序列化：如基本类型、实现了 Parcellable 接口的对象、实现了 Serializable 接口的对象或者是 Android 支持的特殊对象。

(2)ContentProvider:支持跨进程访问。

(3)　广播

(4)共享文件和 SharedPreferences:适合用于数据同步要求不高的进程间通信，要妥善处理并发读/写的问题。

在面对高并发的读/写访问时：

文件支持多个进程同时写，所以高并发情况下读取的内容可能不是最

新的。

SharedPreferences：由于其读写的缓存策略，即在内存中有一份 SharePreferences 文件的缓存，多并发情况下，有很大几率丢失数据，不建议用于跨进程通信。

(5) 基于 Binder 的 Messenger

注意：handler 是跨线程！

①Messenger

Messenger 是轻量级的 IPC 方案，它的底层实现是 AIDL。Messenger 封装了 AIDL，一次处理一个请求，不存在多线程并发的问题。

在不同的进程中共传递 Message 对象(Handler 中的 Messager，因此 Handler 是 Messenger 的基础)。

Messenger 和 Message 实现了 Parcelable 接口。

Messenger 载体：

what、arg1、arg2、Bundle、replyTo、object（只有系统定义的实现了 Parcelable 接口的对象才可以跨进程传递，自定义的实现了 Parcelable 接口的对象不可通过该字段传输）

②原理

使用 Messenger 为服务创建接口，客户端就可利用 Message 对象向服务发送命令。同时客户端也可定义自有 Messenger（在 Message 中可以存放我们需要传递的数据），以便服务回传消息。

因为 Messenger 会在单一线程中创建包含所有请求的队列，以串行的方式处理客户端发来的消息，所以不需要担心线程安全问题。

(6) AIDL（Android Interface Description Language，Android 接口描述语言）

AIDL 是一种 android 内部进程通信接口的描述语言，通过它我们可以定义进程间的通信接口.编译器可以通过扩展名为 aidl 的文件生成一段代码，通过预先定义的接口达到两个进程内部通信进程的目的。

Android 系统中的进程之间不能共享内存，Android 通过 AIDL 来公开服务的接口，将这种可以跨进程访问的服务称为 AIDL 服务。AIDL 可以生成进程间的接口的代码，诸如 service 可能使用的接口。在 Service（服务）的 onBind(Intent intent)方法中返回 mBinder 实现了 aidl 接口的对象

场景:

实现跨进程的方法调用。

想让服务同时处理多个请求，则应该使用 AIDL。

AIDL 文件支持数据类型:

使用这些类型时不需要 import 声明。

Binder 连接池:

创建一个 Service 即可完成多个 AIDL 接口的工作。

建立 AIDL 服务

1) 建立一个扩展名为 aidl 的文件。
2) 如果 aidl 文件的内容是正确的，android studio 会自动生成一个 Java 接口文件（*.java）。
3) 建立一个服务类（Service 的子类）。
4) 实现由 aidl 文件生成的 Java 接口。
5) 在 AndroidManifest.xml 文件中配置 AIDL 服务,尤其要注意的是，<action>标签中 android:name 的属性值就是客户端要引用该服务的 ID，也就是 Intent 类的参数值。

注意:

1) aidl 对应的接口名称必须与 aidl 文件名相同不然无法自动编译。
2) aidl 对应的接口的方法不能加访问权限修饰符，也不能用 final,static。
3) 自定义类型和 AIDL 生成的其它接口类型在 aidl 描述文件中，应该显式 import，即便在该类和定义的包在同一个包中。
4) 在 aidl 文件中所有非 Java 基本类型参数必须加上 in、out、inout 标记，以指明参数是输入参数、输出参数还是输入输出参数。
5) Java 原始类型默认的标记为 in,不能为其它标记。

(7) Socket 和网络通信

Socket 分为流式套接字和用户数据报套接字。

注意：

A.不要在主线程访问网络（Android4.0 以上设备会抛出异常：android.os.NetworkOnMainThreadException）

B.当 Activity 退出时，关闭当前 Socket

18. 多进程场景遇见过么？

19. 谈谈对多进程开发的理解以及多进程应用场景

http://blog.spinytech.com/2016/11/17/android_multiple_process_usage_scenario/

什么情况下需要使用多进程？

常驻后台任务应用:类似音乐类、跑步健身类、手机管家类等长时间需要在后台运行的应用。

这些应用的特点就是，当用户切到别的应用，或者关掉手机屏幕的时候，应用本身的核心模块还在正常运行，提供服务。合理利用多进程，将核心后台服务模块和其他 UI 模块进行分离，保证应用能更稳定的提供服务，从而提升用户体验。

一般会使用两种方式：

1）在新的进程中，启动后台 Service，播放音乐。(一个拿着普通票的妈妈，带着一个拿着中等票的孩子参观)

2) 在新的进程中，启动前台 Service，播放音乐。(一个拿着普通票的妈妈，带着一个 VIP 的孩子去参观)

D 是最智慧的方案.将调度权还给系统，做好自己，维护好整个 Android 生态。,E 是最稳定的方案

多模块应用

多进程还有一种非常有用的场景，就是多模块应用。比如我做的应用大而全，里面肯定会有很多模块，假如有地图模块、大图浏览、自定义 WebView 等等（这些都是吃内存大户），还会有一些诸如下载服务，监控服务等等，一个成熟的应用一定是多模块化的。

1) **解决了 OOM 问题：**首先多进程开发能为应用解决了 OOM 问题，Android 对内存的限制是针对于进程的，这个阈值可以是 48M、24M、16M 等，视机型而定，所以，当我们需要加载大图之类的操作，可以在新的进程中去执行，避免主进程 OOM。

2) **能更有效利用内存：**多进程不光解决 OOM 问题，还能更有效、合理的利用内存。我们可以在适当的时候生成新的进程，在不需要的时候及时杀掉，合理分配，提升用户体验。减少系统被杀掉的风险。

3) **单一进程崩溃不影响整体应用：**多进程还能带来一个好处就是，单一进程崩溃并不影响整体应用的使用。例如我在图片浏览进程打开了一个过大的图片，java heap 申请内存失败，但是不影响

我主进程的使用，而且，还能通过监控进程，将这个错误上报给系统，告知他在什么机型、环境下、产生了什么样的 Bug，提升用户体验。

4) **模块解耦、模块化：**再一个好处就是，当我们的应用开发越来越大，模块越来越多，团队规模也越来越大，协作开发也是个很麻烦的事情。项目解耦，模块化，是这阶段的目标。通过模块解耦，开辟新的进程，独立的 JVM，来达到数据解耦目的。模块之间互不干预，团队并行开发，责任分工也明确。

20. Android 进程分类？

https://blog.csdn.net/zhongshujunqia/article/details/72458271

程序与进程

程序：存储在磁盘上的可运行的代码和数据的集合，是个静态的概念

进程：程序的执行过程，是操作系统进行资源分配的基本单位，是个动态概念

程序由一个或多个相互协作的进程组合而成。

进程的创建

当程序启动运行时，系统就会为之创建相应的进程。在进程当中，调用系统资源，执行程序的逻辑。

进程的销毁

进程什么时候会销毁呢？进程的销毁场景有两种，1.程序不需要继续执行代码，运行结束；2.系统为回收内存，强制销毁。

进程类型

进程优先级从高到低可分为四种：**前台进程、可视进程、服务进程、缓存进程**。（进程被系统强制销毁时，是按照进程的优先级进行的。而进程的优先级主要和应用包含的组件相关。）

(1)**前台进程（foreground process）：**需要用户当前正在进行的操作。一般满足以下条件：

1. 屏幕顶层运行 Activity（处于 onResume()状态），用户正与之交互

2. 有 BroadcastReceiver 正在执行代码

3. 有 Service 在其回调方法（onCreate()、onStart()、onDestroy()）中正在执行代码

这种进程较少，一般来作为最后的手段来回收内存

(2)**可视进程（visible process）：**做用户当前意识到的工作。一般满足以下条件：

1. 屏幕上显示 Activity，但不可操作（处于 onPause()状态）

2. 有 service 通过调用 Service.startForeground()，作为一

个前台服务运行

3. 含有用户意识到的特定的服务，如动态壁纸、输入法等

这些进程很重要，一般不会杀死，除非这样做可以使得所有前台进程存活。

(3)**服务进程（service process）：**含有以 startService()方法启动的 service。虽然该进程用户不直接可见，但是它们一般做一些用户关注的事情（如数据的上传与下载）。

这些进程一般不会杀死，除非系统内存不足以保持前台进程和可视进程的运行。

对于长时间运行的 service（如 30 分钟以上），系统会考虑将之降级为缓存进程，避免长时间运行导致内存泄漏或其他问题，占用过多 RAM 以至于系统无法分配充足资源给缓存进程。

(4)**缓存/后台进程（cached/background process）:**一般来说包含以下条件：

1. 包含多个 Activity 实例，但是都不可见（处于 onStop()且已返回）。

系统如有内存需要，可随意杀死。

21. 进程调度

无论是在批处理系统还是分时系统中，用户进程数一般都多于处理机数、这将导致它们互相争夺处理机。另外，系统进程也同样需要使用处理机。

这就要求进程调度程序按一定的策略，动态地把处理机分配给处于就绪队列中的某一个进程，以使之执行。

22. 进程间通信的方式？

https://www.jianshu.com/p/ce1e35c84134

进程的特点：

- 进程是系统资源和分配的基本单位，而线程是调度的基本单位。
- 每个进程都有自己独立的资源和内存空间
- 其它进程不能任意访问当前进程的内存和资源
- 系统给每个进程分配的内存会有限制

IPC：InterProcess Communication，即进程间通信。

Android 中提供了进程间通信的三种方法：

1）系统实现。

2）AIDL（Android Interface Definition Language，Android 接口定义语言）：大部分应用程序不应该使用 AIDL 去创建一个绑定服务，因为它需要多线程能力，并可能导致一个更复杂的实现。

3）Messenger：利用 Handler 实现。（适用于多进程、单线程，不需要考虑线程安全），其底层基于 AIDL。

使用 Messenger：

如需让服务与远程进程通信，则可使用 Messenger 为服务提供接口。

定义一个 MessengerService 继承自 Service，并在 AndroidManifest.xml 中声明并给一个进程名，使该服务成为一个单独的进程。

Messenger 的使用方法：

1) 服务实现一个 Handler，由其接收来自客户端的每个调用的回调。
2) Handler 用于创建 Messenger 对象（对 Handler 的引用）。
3) Messenger 创建一个 IBinder，服务通过 onBind()使其返回客户端。
4) 客户端使用 IBinder 将 Messenger(引用服务的 Handler)实例化，然后使用后者将 Message 对象发送给服务。
5) 服务在其 Handler 中（具体地讲，是在 handleMessage()方法中）接收每个 Message。

使用 AIDL：

AIDL 是一种接口描述语言，通常用于进程间通信。

使用 AIDL 的步骤：

1) 创建 AIDL，在 main 下新建一个文件夹 aidl，然后在 aidl 下新建 AIDL 文件，这时系统会自动为该文件创建一个包名。

2) 在 java 下新建一个类 AIDLService 继承自 Service。代码如下：

3) 在 AndroidManifest.xml 中注册，并给一个进程名，是该服务成为一个独立的进程。

4) 在 MainActivity 中进行与 AIDLService 之间的进程间通信。