5.Handler

5.1.Handler的实现原理

从四个方面看Handler、Message、MessageQueue 和 Looper Handler:负责消息的发送和处理 Message:消息对象，类似于链表的一个结点; MessageQueue:消息队列，用于存放消息对象的数据结构; Looper:消息队列的处理者（ 用于轮询消息队列的消息对象) Handler发送消息时调用MessageQueue的enqueueMessage插入一条信息到MessageQueue,Looper不断轮询调用MeaasgaQueue的next方法 如果发现message就调用handler的dispatchMessage，dispatchMessage被成功调用，接着调用handlerMessage()。

5.2.子线程中能不能直接new一个Handler,为什么主线程可以

主线程的Looper第一次调用loop方法,什么时候,哪个类 不能，因为Handler 的构造方法中，会通过Looper.myLooper()获取looper对象，如果为空，则抛出异常， 主线程则因为已在入口处ActivityThread的main方法中通过 Looper.prepareMainLooper()获取到这个对象， 并通过 Looper.loop()开启循环，在子线程中若要使用handler，可先通过Loop.prepare获取到looper对象，并使用Looper.loop()开启循环

5.3.Handler导致的内存泄露原因及其解决方案

原因:1.Java中非静态内部类和匿名内部类都会隐式持有当前类的外部引用 2.我们在Activity中使用非静态内部类初始化了一个Handler,此Handler就会持有当前Activity的引用。 3.我们想要一个对象被回收，那么前提它不被任何其它对象持有引用，所以当我们Activity页面关闭之后,存在引用关系："未被处理 / 正处理的消息 -> Handler实例 -> 外部类",如果在Handler消息队列 还有未处理的消息 / 正在处理消息时 导致Activity不会被回收，从而造成内存泄漏 解决方案: 1.将Handler的子类设置成 静态内部类,使用WeakReference弱引用持有Activity实例 2.当外部类结束生命周期时，清空Handler内消息队列

5.4.一个线程可以有几个Handler,几个Looper,几个MessageQueue对象

一个线程可以有多个Handler,只有一个Looper对象,只有一个MessageQueue对象。Looper.prepare()函数中知道，。在Looper的prepare方法中创建了Looper对象，并放入到ThreadLocal中，并通过ThreadLocal来获取looper的对象, ThreadLocal的内部维护了一个ThreadLocalMap类,ThreadLocalMap是以当前thread做为key的,因此可以得知，一个线程最多只能有一个Looper对象， 在Looper的构造方法中创建了MessageQueue对象，并赋值给mQueue字段。因为Looper对象只有一个，那么Messagequeue对象肯定只有一个。

5.5.Message对象创建的方式有哪些 & 区别

Message.obtain()怎么维护消息池的 1.Message msg = new Message(); 每次需要Message对象的时候都创建一个新的对象，每次都要去堆内存开辟对象存储空间 2.Message msg = Message.obtain(); obtainMessage能避免重复Message创建对象。它先判断消息池是不是为空，如果非空的话就从消息池表头的Message取走,再把表头指向next。 如果消息池为空的话说明还没有Message被放进去，那么就new出来一个Message对象。消息池使用Message 链表结构实现，消息池默认最大值 50。消息在loop中被handler分发消费之后会执行回收的操作，将该消息内部数据清空并添加到消息链表的表头。 3.Message msg = handler.obtainMessage(); 其内部也是调用的obtain()方法

5.6.Handler 有哪些发送消息的方法

sendMessage(Message msg) sendMessageDelayed(Message msg, long uptimeMillis) post(Runnable r) postDelayed(Runnable r, long uptimeMillis) sendMessageAtTime(Message msg,long when)

5.7.Handler的post与sendMessage的区别和应用场景

1.源码

sendMessage sendMessage-sendMessageAtTime-enqueueMessage。

post

sendMessage-getPostMessage-sendMessageAtTime-enqueueMessage getPostMessage会先生成一个Messgae，并且把runnable赋值给message的callback

2.Looper->dispatchMessage处理时

public void dispatchMessage(@NonNull Message msg) { if (msg.callback != null) { handleCallback(msg); } else { if (mCallback != null) { if (mCallback.handleMessage(msg)) { return; } } handleMessage(msg); } } dispatchMessage方法中直接执行post中的runnable方法。

而sendMessage中如果mCallback不为null就会调用mCallback.handleMessage(msg)方法，如果handler内的callback不为空，执行mCallback.handleMessage(msg)这个处理消息并判断返回是否为true，如果返回true，消息处理结束，如果返回false,handleMessage(msg)处理。否则会直接调用handleMessage方法。

post方法和handleMessage方法的不同在于，区别就是调用post方法的消息是在post传递的Runnable对象的run方法中处理，而调用sendMessage方法需要重写handleMessage方法或者给handler设置callback，在callback的handleMessage中处理并返回true

应用场景

post一般用于单个场景 比如单一的倒计时弹框功能 sendMessage的回调需要去实现handleMessage Message则做为参数 用于多判断条件的场景。

5.8.handler postDealy后消息队列有什么变化，假设先 postDelay 10s, 再 postDelay 1s, 怎么处理这2条消息sendMessageDelayed-sendMessageAtTime-sendMessage

postDelayed传入的时间，会和当前的时间SystemClock.uptimeMillis()做加和,而不是单纯的只是用延时时间。延时消息会和当前消息队列里的消息头的执行时间做对比，如果比头的时间靠前，则会做为新的消息头，不然则会从消息头开始向后遍历，找到合适的位置插入延时消息。 postDelay()一个10秒钟的Runnable A、消息进队，MessageQueue调用nativePollOnce()阻塞，Looper阻塞； 紧接着post()一个Runnable B、消息进队，判断现在A时间还没到、正在阻塞，把B插入消息队列的头部（A的前面），然后调用nativeWake()方法唤醒线程；MessageQueue.next()方法被唤醒后，重新开始读取消息链表，第一个消息B无延时，直接返回给Looper； Looper处理完这个消息再次调用next()方法，MessageQueue继续读取消息链表，第二个消息A还没到时间，计算一下剩余时间（假如还剩9秒）继续调用nativePollOnce()阻塞； 直到阻塞时间到或者下一次有Message进队；

5.9.MessageQueue是什么数据结构

内部存储结构并不是真正的队列，而是采用单链表的数据结构来存储消息列表 这点和传统的队列有点不一样，主要区别在于Android的这个队列中的消息是按照时间先后顺序来存储的，时间较早的消息，越靠近队头。 当然，我们也可以理解成，它是先进先出的，只是这里的先依据的不是谁先入队，而是消息待发送的时间

5.10.Handler怎么做到的一个线程对应一个Looper，如何保证只有一个MessageQueue ThreadLocal在Handler机制中的作用

设计的初衷是为了解决多线程编程中的资源共享问题， synchronized采取的是“以时间换空间”的策略，本质上是对关键资源上锁，让大家排队操作。 而ThreadLocal采取的是“以空间换时间”的思路， 它一个线程内部的数据存储类，通过它可以在制定的线程中存储数据，数据存储以后，只有在指定线程中可以获取到存储的数据， 对于其他线程就获取不到数据，可以保证本线程任何时间操纵的都是同一个对象。比如对于Handler，它要获取当前线程的Looper,很显然Looper的作用域就是线程，并且不同线程具有不同的Looper。 ThreadLocal本质是操作线程中ThreadLocalMap来实现本地线程变量的存储的 ThreadLocalMap是采用数组的方式来存储数据，其中key(弱引用)指向当前ThreadLocal对象，value为设的值 通过ThreadLocal计算出Hash key，通过这个哈 ThreadLocal对象，value为设的值

5.11.HandlerThread是什么 & 好处 &原理 & 使用场景

HHandlerThread本质上是一个线程类，它继承了Thread； HandlerThread有自己的内部Looper对象，通过Looper.loop()进行looper循环； 通过获取HandlerThread的looper对象传递给Handler对象，然后在handleMessage()方法中执行异步任务；

优势: 1.将loop运行在子线程中处理,减轻了主线程的压力,使主线程更流畅,有自己的消息队列,不会干扰UI线程 2.串行执行,开启一个线程起到多个线程的作用

劣势: 1.由于每一个任务队列逐步执行,一旦队列耗时过长,消息延时 2.对于IO等操作,线程等待,不能并发

我们可以使用HandlerThread处理本地IO读写操作（数据库，文件），因为本地IO操作大多数的耗时属于毫秒级别，对于单线程 + 异步队列的形式 不会产生较大的阻塞

5.12.IdleHandler及其使用场景

Handler 机制提供的一种，可以在 Looper 事件循环的过程中，当出现空闲的时候，允许我们执行任务的一种机制. IdleHandler在looper里面的message处理完了的时候去调用 怎么使用 IdleHandler 被定义在 MessageQueue 中，它是一个接口. 定义时需要实现其 queueIdle() 方法。返回值为 true 表示是一个持久的 IdleHandler 会重复使用，返回 false 表示是一个一次性的 IdleHandler。 IdleHandler 被 MessageQueue 管理，对应的提供了 addIdleHandler() 和 removeIdleHandler() 方法。将其存入 mIdleHandle addIdleHandler() 和 removeIdleHandler() 方法。将其存入 mIdleHandlers 这个 ArrayList 中。 什么时候掉用 就在MessageQueue的next方法里面。 MessageQueue 为空，没有 Message； MessageQueue 中最近待处理的 Message，是一个延迟消息（when>currentTime），需要滞后执行； 使用场景 1.Activity启动优化：onCreate，onStart，onResume中耗时较短但非必要的代码可以放到IdleHandler中执行，减少启动时间

2.想要在一个View绘制完成之后添加其他依赖于这个View的View，当然这个用View#post()也能实现，区别就是前者会在消息队列空闲时执行 优化页面的启动,较复杂的view填充 填充里面的数据界面view绘制之前的话，就会出现以上的效果了，view先是白的，再出现. app的进程其实是ActivityThread,performResumeActivity先回调onResume ， 之后 执行view绘制的measure, layout, draw,也就是说onResume的方法是在绘制之前，在onResume中做一些耗时操作都会影响启动时间 把在onResume以及其之前的调用的但非必须的事件（如某些界面View的绘制）挪出来找一个时机（即绘制完成以后）去调用即可。

5.13.消息屏障，同步屏障机制what

同步屏障只在Looper死循环获取待处理消息时才会起作用，也就是说同步屏障在MessageQueue.next函数中发挥着作用。

在next()方法中，有一个屏障的概念(message.target ==null为屏障消息), 遇到target为null的Message，说明是同步屏障，循环遍历找出一条异步消息，然后处理。 在同步屏障没移除前，只会处理异步消息，处理完所有的异步消息后，就会处于堵塞 当出现屏障的时候，会滤过同步消息，而是直接获取其中的异步消息并返回, 就是这样来实现「异步消息优先执行」的功能

how

1、Handler构造方法中传入async参数，设置为true，使用此Handler添加的Message都是异步的； 2、创建Message对象时，直接调用setAsynchronous(true) 3.removeSyncBarrier() 移除同步屏障：

应用

在 View 更新时，draw、requestLayout、invalidate 等很多地方都调用了ViewRootImpl#scheduleTraversals(Android应用框架中为了更快的响应UI刷新事件在ViewRootImpl.scheduleTraversals中使用了同步屏障

5.14.子线程能不能更新UI

刷新UI， 都会调用到ViewRootImpl.Android每次刷新UI的时候，最终根布局ViewRootImpl.checkThread()来检验线程是否是View的创建线程。 ViewRootImpl创建的第一个地方，从Acitivity声明周期handleResumeActivity会被优先调用到，也就是说在OnResume后ViewRootImpl就被创建，这个时候无法在在子线程中访问UI了，上面子线程延迟了一会，handleResumeActivity已经被调用了，所以发生了崩溃 不延迟在creae里直接设置不会崩溃 线程更新UI也行，但是只能更新自己创建的View

5.15.为什么Android系统不建议子线程访问UI

在android中子线程可以有好多个，但是如果每个线程都可以对ui进行访问，我们的界面可能就会变得混乱不堪，这样多个线程操作同一资源就会造成线程安全问题，当然，需要解决线程安全问题的时候，我们第一想到的可能就是加锁，但是加锁会降低运行效率，所以android出于性能的考虑，并没有使用加锁来进行ui操作的控制。

5.16.Android中为什么主线程不会因为Looper.loop()里的死循环卡死？

MessageQueue#next 在没有消息的时候会阻塞，如何恢复？

他不阻塞的原因是epoll机制，他是linux里面的，在native层会有一个读取端和一个写入端，当有消息发送过来的时候会去唤醒读取端，然后进行消息发送与处理，没消息的时候是处于休眠状态，所以他不会阻塞他。

5.17.Handler消息机制中，一个looper是如何区分多个Handler的当Activity有多个Handler的时候，怎么样区分当前消息由哪个Handler处理处理message的时候怎么知道是去哪个callback处理的

每个Handler会被添加到 Message 的target字段上面，Looper 通过调用 Message.target.handleMessage() 来让 Handler 处理消息。

5.18.Looper.quit/quitSafely的区别

当我们调用Looper的quit方法时，实际上执行了MessageQueue中的removeAllMessagesLocked方法，该方法的作用是把MessageQueue消息池中所有的消息全部清空， 无论是延迟消息（延迟消息是指通过sendMessageDelayed或通过postDelayed等方法发送的需要延迟执行的消息）还是非延迟消息。

当我们调用Looper的quitSafely方法时，实际上执行了MessageQueue中的removeAllFutureMessagesLocked方法，通过名字就可以看出，该方法只会清空MessageQueue消息池中所有的延迟消息，并将消息池中所有的非延迟消息派发出去让Handler去处理，quitSafely相比于quit方法安全之处在于清空消息之前会派发所有的非延迟消息。

5.19.通过Handler如何实现线程的切换

当在A线程中创建handler的时候，同时创建了MessageQueue与Looper，Looper在A线程中调用loop进入一个无限的for循环从MessageQueue中取消息，当B线程调用handler发送一个message的时候，会通过msg.target.dispatchMessage(msg);将message插入到handler对应的MessageQueue中，Looper发现有message插入到MessageQueue中，便取出message执行相应的逻辑，因为Looper.loop()是在A线程中启动的，所以则回到了A线程，达到了从B线程切换到A线程的目的。

5.20.Handler 如何与 Looper 关联的

通过构造方法 mLooper = Looper.myLooper()->sThreadLocal.get()(sThreadLocal.set)

5.21.Looper 如何与 Thread 关联的

Looper 与 Thread 之间是通过 ThreadLocal 关联的，这个可以看 Looper#prepare() 方法 Looper 中有一个 ThreadLocal 类型的 sThreadLocal静态字段，Looper通过它的 get 和 set 方法来赋值和取值。 由于 ThreadLocal是与线程绑定的，所以我们只要把 Looper 与 ThreadLocal 绑定了，那 Looper 和 Thread 也就关联上了

5.22.Looper.loop()源码

for无限循环，阻塞于消息队列的next方法 取出消息后调用msg.target.dispatchMessage(msg)进行消息分发

5.23.MessageQueue的enqueueMessage()方法如何进行线程同步的

就是单链表的插入操作 如果消息队列被阻塞回调用nativeWake去唤醒。 用synchronized代码块去进行同步。

5.24.MessageQueue的next()方法内部原理

next() 是如何处理一般消息的？

next() 是如何处理同步屏障的？

next() 是如何处理延迟消息的？

调用 MessageQueue.next() 方法的时候会调用 Native 层的 nativePollOnce() 方法进行精准时间的阻塞。在 Native 层，将进入 pullInner() 方法，使用 epoll_wait 阻塞等待以读取管道的通知。如果没有从 Native 层得到消息，那么这个方法就不会返回。此时主线程会释放 CPU 资源进入休眠状态。

5.25.子线程中是否可以用MainLooper去创建Handler，Looper和Handler是否一定处于一个线程

可以的。 子线程中Handler handler = new Handler(Looper.getMainLooper());，此时两者就不在一个线程中

5.26.ANR和Handler的联系

Handler是线程间通讯的机制，Android中，网络访问、文件处理等耗时操作必须放到子线程中去执行，否则将会造成ANR异常。 ANR异常：Application Not Response 应用程序无响应 产生ANR异常的原因：在主线程执行了耗时操作，对Activity来说，主线程阻塞5秒将造成ANR异常，对BroadcastReceiver来说，主线程阻塞10秒将会造成ANR异常。 解决ANR异常的方法：耗时操作都在子线程中去执行 但是，Android不允许在子线程去修改UI，可我们又有在子线程去修改UI的需求，因此需要借助Handler。