面试题总结

Handler是一个比较重要的东西，所以把网上发的Handler中的面试题总结了一下，这些面试题没问题的话，Handler源码相关的内容就应该没问题了，有空的话会再写一个Handler源码篇。

[Q1：用一句话概括Handler，并简述其原理。]

> Handler是Android系统的根本，在Android应用被启动的时候，会分配一个单独的虚拟机，虚拟机会执行ActivityThread中的main方法，在main方法中对主线程Looper进行了初始化，也就是几乎所有代码都执行在Handler内部。Handler也可以作为主线程和子线程通讯的桥梁。

> Handler通过sendMessage发送消息，将消息放入MessageQueue中，在MessageQueue中通过时间的维度来进行排序，Looper通过调用loop方法不断的从MessageQueue中获取消息，执行Handler的dispatchMessage，最后调用handleMessage方法。

[Q2：为什么系统不建议在子线程访问UI？（为什么不能在子线程更新UI？）]

> 在某些情况下，在子线程中是可以更新UI的。但是在ViewRootImpl中对UI操作进行了checkThread，但是我们在OnCreate和onResume中可以使用子线程更新UI，由于我们在ActivityThread中的performResumeActivity方法中通过addView创建了ViewRootImpl，这个行为是在onResume之后调用的，所以在OnCreate和onResume可以进行更新UI。

> 但是我们不能在子线程中更新UI，因为如果添加了耗时操作之后，一旦ViewRootImpl被创建将会抛出异常。一旦在子线程中更新UI，容易产生并发问题。

[Q3：一个Thread可以有几个Looper？几个Handler？]

> 一个线程只能有一个Looper，可以有多个Handler，在线程中我们需要调用Looper.perpare,他会创建一个Looper并且将Looper保存在ThreadLocal中，每个线程都有一个LocalThreadMap，会将Looper保存在对应线程中的LocalThreadMap，key为ThreadLocal，value为Looper

[Q4：可以在子线程直接new一个Handler吗？那该怎么做？]

> 可以在子线程中创建Handler，我们需要调用Looper.perpare和Looper.loop方法。或者通过获取主线程的looper来创建Handler

[Q5：Message可以如何创建？哪种效果更好，为什么？]

> Message.obtain来创建Message。这样会复用之前的Message的内存，不会频繁的创建对象，导致内存抖动。

[Q6：主线程中Looper的轮询死循环为何没有阻塞主线程？]

> Looper轮询是死循环，但是当没有消息的时候他会block，ANR是当我们处理点击事件的时候5s内没有响应，我们在处理点击事件的时候也是用的Handler，所以一定会有消息执行，并且ANR也会发送Handler消息，所以不会阻塞主线程。

[Q7：使用Hanlder的postDealy()后消息队列会发生什么变化？]

> Handler发送消息到消息队列，消息队列是一个时间优先级队列，内部是一个单向链表。发动postDelay之后会将该消息进行时间排序存放到消息队列中

[Q8：点击页面上的按钮后更新TextView的内容，谈谈你的理解？（阿里面试题）]

> 点击按钮的时候会发送消息到Handler，但是为了保证优先执行，会加一个标记异步，同时会发送一个target为null的消息，这样在使用消息队列的next获取消息的时候，如果发现消息的target为null，那么会遍历消息队列将有异步标记的消息获取出来优先执行，执行完之后会将target为null的消息移除。(同步屏障)

[Q9：生产者-消费者设计模式懂不？]

> 举个例子，面包店厨师不断在制作面包，客人来了之后就购买面包，这就是一个典型的生产者消费者设计模式。但是需要注意的是如果消费者消费能力大于生产者，或者生产者生产能力大于消费者，需要一个限制，在java里有一个blockingQueue。当目前容器内没有东西的时候，消费者来消费的时候会被阻塞，当容器满了的时候也会被阻塞。Handler.sendMessage相当于一个生产者,MessageQueue相当于容器，Looper相当于消费者。

[Q10：Handler是如何完成子线程和主线程通信的？]

> 在主线程中创建Handler，在子线程中发送消息，放入到MessageQueue中,通过Looper.loop取出消息进行执行handleMessage，由于looper我们是在主线程初始化的，在初始化looper的时候会创建消息队列，所以消息是在主线程被执行的。

[Q11：关于ThreadLocal，谈谈你的理解？]

> ThreadLocal类似于每个线程有一个单独的内存空间，不共享，ThreadLocal在set的时候会将数据存入对应线程的ThreadLocalMap中，key=ThreadLocal，value=值

[Q12：享元设计模式有用到吗？]

> 享元设计模式就是重复利用内存空间，减少对象的创建，Message中使用到了享元设计模式。内部维护了一个链表，并且最大长度是50，当消息处理完之后会将消息内的属性设置为空，并且插入到链表的头部，使用obtain创建的Message会从头部获取空的Message

[Q13: Handler内存泄漏问题及解决方案]

> 内部类持有外部类的引用导致了内存泄漏，如果Activity退出的时候，MessageQueue中还有一个Message没有执行，这个Message持有了Handler的引用，而Handler持有了Activity的引用，导致Activity无法被回收，导致内存泄漏。使用static关键字修饰，在onDestory的时候将消息清除。

[Q14: Handler异步消息处理（HandlerThread）]

> 内部使用了Handler+Thread，并且处理了getLooper的并发问题。如果获取Looper的时候发现Looper还没创建，则wait，等待looper创建了之后在notify

[Q15: 子线程中维护的Looper，消息队列无消息的时候处理方案是什么？有什么用？]

> 子线程中创建了Looper，当没有消息的时候子线程将会被block，无法被回收，所以我们需要手动调用quit方法将消息删除并且唤醒looper，然后next方法返回null退出loop

[Q16: 既然可以存在多个Handler往MessageQueue中添加数据(发消息是各个handler可能处于不同线程)，那他内部是如何确保线程安全的？]

> 在添加数据和执行next的时候都加了this锁，这样可以保证添加的位置是正确的，获取的也会是最前面的。

[Q17: 关于IntentService，谈谈你的理解]

> HandlerThread+Service实现，可以实现Service在子线程中执行耗时操作，并且执行完耗时操作时候会将自己stop。

[Q18: Glide是如何维护生命周期的？]
一般想问的应该都是这里

```
@NonNull
 private RequestManagerFragment getRequestManagerFragment(
     @NonNull final android.app.FragmentManager fm,
     @Nullable android.app.Fragment parentHint,
```

```
      boolean isParentVisible) {
    RequestManagerFragment current = (RequestManagerFragment)
fm.findFragmentByTag(FRAGMENT_TAG);
    if (current == null) {
      current = pendingRequestManagerFragments.get(fm);
      if (current == null) {
        current = new RequestManagerFragment();
        current.setParentFragmentHint(parentHint);
        if (isParentVisible) {
          current.getGlideLifecycle().onStart();
        }
        pendingRequestManagerFragments.put(fm, current);
        fm.beginTransaction().add(current,
FRAGMENT_TAG).commitAllowingStateLoss();
        handler.obtainMessage(ID_REMOVE_FRAGMENT_MANAGER, fm).sendToTarget();
      }
    }
    return current;
  }
```

> 1.为什么会判断两次null，再多次调用with的时候，commitAllowingStateLoss会被执行两次，所以我们需要使用一个map集合来判断，如果map中已经有了证明已经添加过了
> 2.handler.obtainMessage(ID_REMOVE_FRAGMENT_MANAGER, fm).sendToTarget();我们需要将map里面的记录删除。