2018

Activity lifeCycle VS view lifeCycle Handler

2018.04.12

哈哈

Handler

```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d( tag: "zd",  msg: "main activity on create");
    setContentView(R.layout.activity_main);
    mTv = findViewById(R.id.id_btn);
    mHandler.post(new Runnable() {
        @Override
        public void run() { Log.d( tag: "zd",  msg: "handler onCreate getWidth : " + mTv.getWidth()); }
    });

    mTv.post(new Runnable() {
        @Override
        public void run() { Log.d( tag: "zd",  msg: "view post getWidth: " + mTv.getWidth()); }
    });
}

@Override
protected void onResume() {
    super.onResume();
    Log.d( tag: "zd",  msg: "main activity on resume");
    mHandler.post(new Runnable() {
        @Override
        public void run() { Log.d( tag: "zd",  msg: "handler onResume getWidth : " + mTv.getWidth()); }
    });
}
```

```java
@Override
protected void onAttachedToWindow() {
    super.onAttachedToWindow();
    Log.d( tag: "zd", msg: "onAttachedToWindow()");
}

@Override
protected void onDetachedFromWindow() {
    super.onDetachedFromWindow();
    Log.d( tag: "zd", msg: "onDetachedFromWindow()");
}

@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    Log.d( tag: "zd",  msg: "View onMeasure");
}

@Override
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
    super.onLayout(changed, left, top, right, bottom);
    Log.d( tag: "zd",  msg: "View onLayout");
}
```

01 handler.post() VS view.post()

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d( tag: "zd",  msg: "main activity on create");
    setContentView(R.layout.activity_main);
    mTv = findViewById(R.id.id_btn);
    mHandler.post(new Runnable() {
        @Override
        public void run() {
            Log.d( tag: "zd",  msg: "handler onCreate getWidth : " + mTv.getWidth());
        }
    });

    mTv.post(new Runnable() {
        @Override
        public void run() {
            Log.d( tag: "zd",  msg: "view post getWidth: " + mTv.getWidth());
        }
    });
}

@Override
protected void onResume() {
    super.onResume();
    Log.d( tag: "zd",  msg: "main activity on resume");
    mHandler.post(new Runnable() {
        @Override
        public void run() {
            Log.d( tag: "zd",  msg: "handler onResume getWidth : " + mTv.getWidth());
        }
    });
}
```

```
main activity on create
main activity on resume
handler onCreate getWidth : 0
handler onResume getWidth : 0
View onMeasure
View onMeasure
View onLayout
view post getWidth: 1440
```

```
onCreate()
activity onResume()
onAttachedToWindow()
View onMeasure
View onMeasure
View onLayout
activity onDestroy() before
activity onDestroy() after
onDetachedFromWindow()
```

01 原因分析

```
{
    ActivityThread : performLaunchActivity()
                          -> handleResumeActivity();
                            -> wm.addView(decor , layoutParams)
    ........
     ViewRootImpl :  setView()
                          -> requestLayout()
                            -> scheduleTraversals()
                              -> doTraversals()
                                 -> postCallBack(mTraversalRunnable)
                                   -> performTraversal() ： 注：这里是 handler post() 出来的.
     ...................
     ViewRootmpl : performMeasure()
     ..............
      ViewRootmpl :  performLayout()
     ..............
      ViewRootmpl  :  performDraw()
}
```

结论

Activity 、Handler.post()

Activity onCreate(), onStart(),onResume() 在同一个 Message 里依次得到执行。 ActvityThread @ performLaunchActivity()

onResume() 完成后才会去 View.onMeasure () , onLayout() ...

由于 performTraversal() 是 handler post() 出去的， 所以在 onCreate() , onResume() 时并拿不到宽高。

01 原因分析

```
{
    View: post(){
        if (attachInfo != null) {
            return attachInfo.mHandler.post(action);
        }
        getRunQueue().post(action);     ： 关键在于 attachInfo 什么时候赋值
    }

    ActivityThread @ handleResumeActivity()
    ...................
    ViewRootmpl : performTraversal()
        ->  host.dispatchAttachedToWindow(mAttachInfo, 0);   : host 为 decorView
          -> getRunQueue().executeActions(mAttachInfo.mHandler);   // HandleActionQueue
        此处将 getRunQueue() 里面的runnable 拿出来，重新抛到主线程 Looper 。
    }
}
```

结论

View.post()

View.post() 分两种情况，如果 attachToWindow() , 那么直接 post() 出去，否则先暂存起来。

在performTraversals() 中， 会给 attachInfo 赋值，并且把暂存的任务拿出来直接 post() 出去。

由于 performTraversals() 完成后就可以拿到宽高，所以，当post() 出去以后，可以正常拿到宽高。

01 HandlerActionQueue

```
// view.post()  -> postDelayed(action , 0)

public void postDelayed(Runnable action, long delayMillis) {
      final HandlerAction handlerAction = new HandlerAction(action, delayMillis);

      synchronized (this) {
         if (mActions == null) {
            mActions = new HandlerAction[4];
         }
         mActions = GrowingArrayUtils.append(mActions, mCount, handlerAction);
         mCount++;
      }
   }
```

```
// 真正执行的地方
public void executeActions(Handler handler) {
      synchronized (this) {
         final HandlerAction[] actions = mActions;
         for (int i = 0, count = mCount; i < count; i++) {
            final HandlerAction handlerAction = actions[i];
            handler.postDelayed(handlerAction.action,
handlerAction.delay);
         }

         mActions = null;
         mCount = 0;
      }
   }
```

01 思考

如果在 onCreate 中通过 Handler 拿到宽高 ？

```
mTv.post(new Runnable() {
        @Override
        public void run() {
            Log.d("zd","mtv.post , textView width : " +          mTv.getWidth());
        }
    });

    Looper.myQueue().addIdleHandler(new MessageQueue.IdleHandler() {
        @Override
        public boolean queueIdle() {
            Log.d("zd","idle handler textView width : " + mTv.getWidth());
            return false;
        }
    });
```

结论

MessageQueue.addIdleHandler()

```
View onMeasure
View onMeasure
View onLayout
mtv.post , textView width : 1440
idle handler textView width : 1440
```

01 MessageQueue.addIdleHandler()

```
/**
    * Add a new {@link IdleHandler} to this message queue.  This may be
    * removed automatically for you by returning false from
    * {@link IdleHandler#queueIdle IdleHandler.queueIdle()} when it is
    * invoked, or explicitly removing it with {@link #removeIdleHandler}.
    *
    * <p>This method is safe to call from any thread.
    *
    * @param handler The IdleHandler to be added.
    */
   public void addIdleHandler(@NonNull IdleHandler handler) {
       if (handler == null) {
           throw new NullPointerException("Can't add a null IdleHandler");
       }
       synchronized (this) {
           mIdleHandlers.add(handler);
       }
   }
```

```
/**
    * Callback interface for discovering when a thread is going to block
    * waiting for more messages.
    */
   public static interface IdleHandler {
       /**
        * Called when the message queue has run out of messages and will
now
        * wait for more.  Return true to keep your idle handler active, false
        * to have it removed.  This may be called if there are still messages
        * pending in the queue, but they are all scheduled to be dispatched
        * after the current time.
        */
       boolean queueIdle();
   }
```

01 MessageQueue.addIdleHandler()

```
MessageQueue@next{→
        .........
        Try to retrieve the next message , Return if found
        .........
        do {
            prevMsg = msg;
            msg = msg.next;
        } while (msg != null && !msg.isAsynchronous());
        .........
        return if found message
        ..........

}
```

```
MessageQueue@next{→
        // Run the idle handlers.
        // We only ever reach this code block during the first iteration.
        for (int i = 0; i < pendingIdleHandlerCount; i++) {
            final IdleHandler idler = mPendingIdleHandlers[i];
            try {
                keep = idler.queueIdle();
            } catch (Throwable t) {
                Log.wtf(TAG, "IdleHandler threw exception", t);
            }
        }
}
```

01 使用场景 ： LeakCanary

```
{
    ..
    register activity lifeCycle callBack
    ..
    // 重点是执行 Gc 的时机
    run Gc when activity onDestroy()
        -> waitForIdle()
            -> Looper.myQueue().addIdleHandler()
                -> ensureGone()
                    -> GcTrigger@runGc()
    ..
    start analyizing
    ...
}
```

结论

LeakCanary 检测内存泄漏时机

在 onDestroy() 之后，会主动回调。

因为是 addIdleHandler() ， 所以必须等到 MessageQueue.next() 拿不到数据的时候，才会去执行 queueIdle();

所以如果在 onDestroy() 中，通过handler.post() 一个或多个 runnable , 是不会被判定为内存泄漏的。

但是如果是 postDelay() 的话，就会有可能判定为内存泄漏。