链滴

android 的 IdleHandler 你了解嘛？？

作者：yc654084303

原文链接：https://ld246.com/article/1571888143440

来源网站：链滴

在Android中，Handler是我们最常用的消息处理类，也是系统中非常重要的类，所以它的原理我们应该十分的清楚了解，所以Handler，Looper，MessageQueue成了我们必须学习和理解的东西，常我们都通过Handler向Looper包含的MessageQueue投递Message，比如这样：

```
new Handler(Looper.getMainLooper()).post(new Runnable() {
@Override  public  void  run() {/do something }
});
```

但是其中有一个接口很少被提及和使用，那就是IdleHandler,其实它内部的IdleHandler接口有很多有的用法，首先看看它的定义：

```
/**
 * 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();
 }
```

简而言之，就是在looper里面的message暂时处理完了，这个时候会回调这个接口，返回false，那就会移除它，返回true就会在下次message处理完了的时候继续回调

下面我们来看下IdleHandler在Handler空闲时执行案例代码：

```
public class MainActivity extends Activity {
  private Handler mHandler;

    private int mWhat = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mHandler = new Handler(){

            @Override
            public void handleMessage(Message msg) {
                Log.d("main", "我在执行，你想回来，我用平底锅打飞你！");
                try {
                    Thread.sleep(1500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                super.handleMessage(msg);
            }

        };
```

```
        Button btn = (Button) findViewById(R.id.but);
        btn.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                for(int i=0;i<10;i++){
                    mHandler.sendEmptyMessage(mWhat);
                }
            }
        });
    }



    @Override
    protected void onResume() {
        super.onResume();

        Looper.myQueue().addIdleHandler(new MyIdleOnce());
        Looper.myQueue().addIdleHandler(new MyIdleKeep());

    }

    class MyIdleKeep implements MessageQueue.IdleHandler{
        /**
         *返回值为true，则保持此Idle一直在Handler中，否则，执行一次后就从Handler线程中remov
掉。
         */
        @Override
        public boolean queueIdle() {
            Log.d("main","我是空闲线程,我还会回来的！");
            return true;
        }

    }

    class MyIdleOnce implements MessageQueue.IdleHandler{

        @Override
        public boolean queueIdle() {
            Log.d("main","我是初恋，我只在你的生命中出现一次，我发誓，你会想我的！");
            return false;
        }

    }
}
```

此Activity有两个IdleHandler消息，我们执行此Activity时，MyIdleKeep消息和MyIdleOnce会依次行。如果IdleHandler的queueIdle方法返回false，那么IdleHandler执行完，就会从IdleHandler移。

Log输出如下:

2019-10-24 11:29:22.439 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还

```
回来的!
2019-10-24 11:29:22.617 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:22.617 8936-8936/com.example.myapplication D/main: 我是初恋，我只在
的生命中出现一次，我发誓，你会想我的!
2019-10-24 11:29:22.617 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:22.680 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:22.798 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:22.798 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:51.819 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:51.819 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:29:51.857 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:29:59.364 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:30:00.866 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:30:02.367 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:30:03.868 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:30:05.369 8936-8936/com.example.myapplication D/main: 我在执行，你想回
，我用平底锅打飞你!
2019-10-24 11:30:06.892 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
2019-10-24 11:30:07.004 8936-8936/com.example.myapplication D/main: 我是空闲线程,我还
回来的!
```

IdleHandler 在 MessageQueue 与 Looper 和 Handler 是什么关系?

- 原理源码分析如下：

```
/**
 * 获取当前线程队列使用Looper.myQueue()，获取主线程队列可用getMainLooper().myQueue()
 */
public final class MessageQueue {
    ......
    /**
     * 当前队列将进入阻塞等待消息时调用该接口回调，即队列空闲
     */
    public static interface IdleHandler {
        /**
         * 返回true就是单次回调后不删除，下次进入空闲时继续回调该方法，false只回调单次。
         */
        boolean queueIdle();
    }
```

```
/**
 * <p>This method is safe to call from any thread.
 * 判断当前队列是不是空闲的，辅助方法
 */
public boolean isIdle() {
    synchronized (this) {
        final long now = SystemClock.uptimeMillis();
        return mMessages == null || now < mMessages.when;
    }
}

/**
 * <p>This method is safe to call from any thread.
 * 添加一个IdleHandler到队列，如果IdleHandler接口方法返回false则执行完会自动删除，
 * 否则需要手动removeIdleHandler。
 */
public void addIdleHandler(@NonNull IdleHandler handler) {
    if (handler == null) {
        throw new NullPointerException("Can't add a null IdleHandler");
    }
    synchronized (this) {
        mIdleHandlers.add(handler);
    }
}

/**
 * <p>This method is safe to call from any thread.
 * 删除一个之前添加的 IdleHandler。
 */
public void removeIdleHandler(@NonNull IdleHandler handler) {
    synchronized (this) {
        mIdleHandlers.remove(handler);
    }
}
......
//Looper的prepare()方法会通过ThreadLocal准备当前线程的MessageQueue实例，
//然后在loop()方法中死循环调用当前队列的next()方法获取Message。
Message next() {
    ......
    for (;;) {
        ......
        nativePollOnce(ptr, nextPollTimeoutMillis);
        synchronized (this) {
            ......
            //把通过addIdleHandler添加的IdleHandler转成数组存起来在mPendingIdleHandlers中
            // If first time idle, then get the number of idlers to run.
            // Idle handles only run if the queue is empty or if the first message
            // in the queue (possibly a barrier) is due to be handled in the future.
            if (pendingIdleHandlerCount < 0
                    && (mMessages == null || now < mMessages.when)) {
                pendingIdleHandlerCount = mIdleHandlers.size();
            }
            if (pendingIdleHandlerCount <= 0) {
                // No idle handlers to run.  Loop and wait some more.
```

```
                mBlocked = true;
                continue;
            }

            if (mPendingIdleHandlers == null) {
                mPendingIdleHandlers = new IdleHandler[Math.max(pendingIdleHandlerCount, 4
];
            }
            mPendingIdleHandlers = mIdleHandlers.toArray(mPendingIdleHandlers);
        }

        // Run the idle handlers.
        // We only ever reach this code block during the first iteration.
        //循环遍历所有IdleHandler
        for (int i = 0; i < pendingIdleHandlerCount; i++) {
            final IdleHandler idler = mPendingIdleHandlers[i];
            mPendingIdleHandlers[i] = null; // release the reference to the handler

            boolean keep = false;
            try {
                //调用IdleHandler接口的queueIdle方法并获取返回值。
                keep = idler.queueIdle();
            } catch (Throwable t) {
                Log.wf(TAG, "IdleHandler threw exception", t);
            }
            //如果IdleHandler接口的queueIdle方法返回false说明只执行一次需要删除。
            if (!keep) {
                synchronized (this) {
                    mIdleHandlers.remove(idler);
                }
            }
        }
        ......
    }
}
```

原文链接：android 的 IdleHandler 你了解嘛？？