Android Handler的原理

1. 简介

众所周知主线程（也叫UI线程，进行UI的更新）是不可以进行任何耗时操作，比如耗时的计算，访问服务器等等。那如果开启子线程进行复杂计算以后想要把结果传递给主线程进行结果的展示（UI更新），这个时候该怎么办比较好呢。

有两种方法，第一种是用AsyncTask，另一种就是要讲的Handler了。当然AsyncTask和Handler用途和原理有所不同，所以应该根据场景选择使用。

Handler是什么？Handler是在Android中用于消息的传递。Android中的主线程会维护一个Looper和MessageQueue，来保证内部的信息传递，当然我们也可以使用它们来完成我们的逻辑。简单说明一下Looper和MessageQueue。

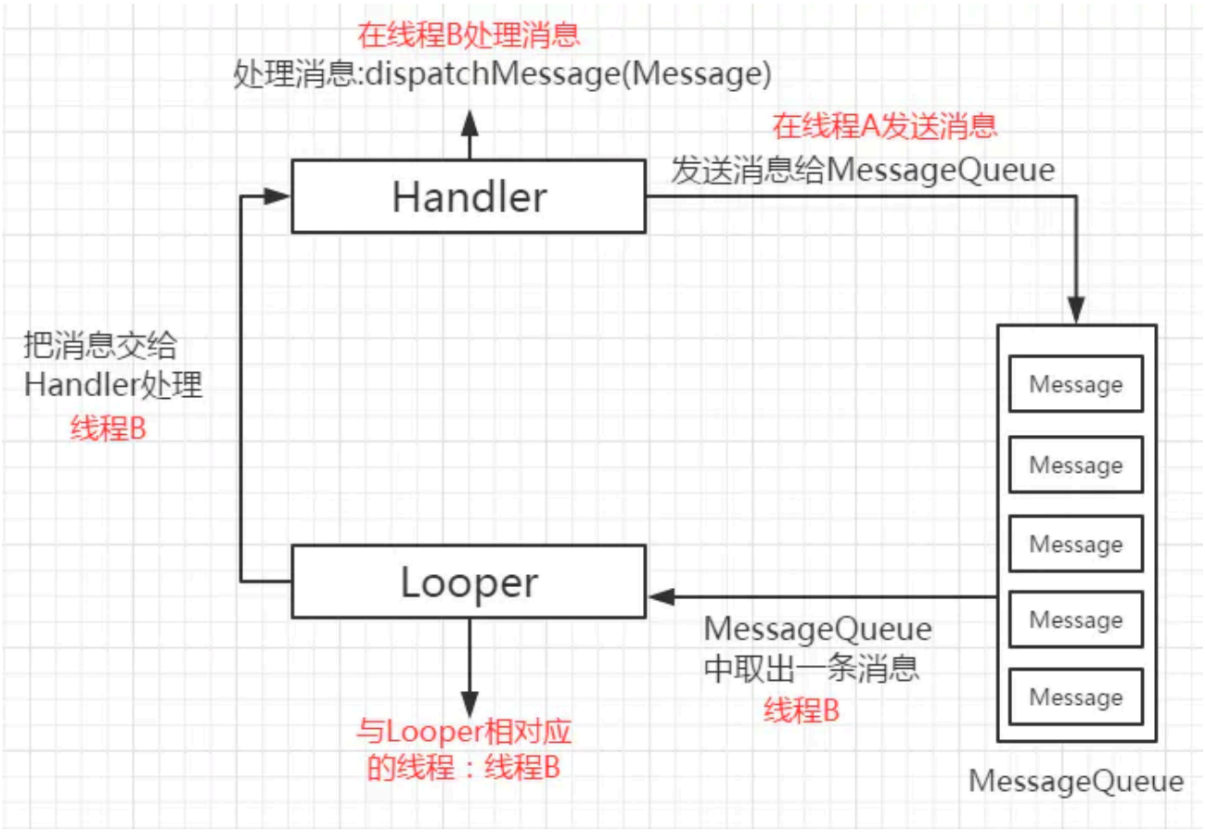
Looper: 它的主要作用是管理维护MessageQueue，从队列中取出消息传递给Handler来处理。
MessageQueue: 它的主要作用是存储消息，当队列为空时进行队列阻塞。

2. 消息传递处理流程

1. Handler在线程A中发出消息，传递到MessageQueue中进行储存。
2. Looper在线程B中取出MessageQueue中的消息。
3. Looper在线程B中把消息传递给Handler。
4. Handler在线程B中处理消息。

一个Looper对应一个线程。如果在主线程中发送的Message存储到共同维护的MessageQueue中，在另一个子线程的Looper进行loop（用于去消息的方法）取消息然后传递给Handler，则Handler会在Looper所在的线程进行消息的处理。

整个流程以及关系如下。



3. 源码分析

1. 首先从Handler的源码开始分析，如下。

```
public Handler() {
    this(null, false);
}

public Handler(Callback callback, boolean async) {
    //获取Looper对象
    mLooper = Looper.myLooper();
    if (mLooper == null) {
        throw new RuntimeException(
            "Can't create handler inside thread that has not called Looper.prepare()");
    }
    //获取Looper对象的mQueue属性，mQueue 就是MessageQueue对象。
    mQueue = mLooper.mQueue;
```

```
        mCallback = callback;
        mAsynchronous = async;
    }
```

Handler中会通过Looper.myLooper() 来获取获取当前线程的Looper，如果为空则会抛出异常。所以在子线程中创建Handler，则首先要通过Looper.prepare() 来创建Looper，再通过Handler传递消息。值得注意的是主线程中已经默认创建了Looper，所以在主线程不要再单独创建Looper，如果创建则会抛出异常。

2. 再看一下Looper.myLooper 和Looper.prepare() 的源码。

```
    static final ThreadLocal<Looper> sThreadLocal = new ThreadLocal<Looper>();

    //创建当前线程的Looper对象
    private static void prepare(boolean quitAllowed) {
        if (sThreadLocal.get() != null) {
            throw new RuntimeException("Only one Looper may be created per thread");
        }
        sThreadLocal.set(new Looper(quitAllowed));
    }

    //获取当前线程的Looper对象
    public static @Nullable Looper myLooper() {
        return sThreadLocal.get();
    }
```

通过prepare() 方法就可以知道每个线程中只能有一个Looper，这也是为什么不能在主线程创建Looper的原因。myLooper() 来获取当前线程的Looper对象。
还有值得注意的细节是，管理Looper的数据类型是ThreadLocal ，它一个线程内部的数据存储类，通过它存储的数据只有在它自己的线程才能获取到，其他线程是获取不到的。与之相对应的是ThreadGlobal 是可以获取全局任意一个线程。所以在这里sThreadLocal.get() 是获取当前线程的数据的意思。

3. 接下来继续看一下Looper。

```
    private Looper(boolean quitAllowed) {
        mQueue = new MessageQueue(quitAllowed); //实例化MessageQueue对象
```

```
        mThread = Thread.currentThread(); //当前线程
    }
```

Looper对象中创建了MessageQueue（消息队列，用于存储消息）对象以及获取了当前的线程。因为Looper的创建是私有的，所以外界想要获取Looper对象只能通过Looper.prepare()方法。

4. 发出消息(sendMessage)

不管是handler.sendMessage(msg) 和handler.post(runnable) 最底层的方法都是下面的enqueueMessage() 方法。通过这个方法把消息存到了MessageQueue中。

```
    private boolean enqueueMessage(MessageQueue queue, Message msg, long
uptimeMillis) {
        msg.target = this; //this是Handler对象。
        if (mAsynchronous) {
            msg.setAsynchronous(true);
        }
        //把消息存到MessageQueue，并返回存储成功失败的结果
        return queue.enqueueMessage(msg, uptimeMillis);
    }
```

5. 取出消息

消息的取出用的是Loooper.loop() 方法。首先看一下代码。

```
public static void loop() {

        final MessageQueue queue = me.mQueue;
        //死循环
        for (;;) {
            //从MessageQueue中取出一条消息
            Message msg = queue.next();
            if (msg == null) {
                // 如果没有消息了，直接跳出循环
                // 但是正常情况下这段并不会被调用
                return;
            }
```

```
            //把消息交给Handler处理。
            msg.target.dispatchMessage(msg);
        }
    }
```

会发现上面代码中有一个死循环，通过这个死循环不断的从MessageQueue中取出消息。当queue.next() 没有获取到消息则会被阻塞，知道有消息被存放到MessageQueue中。当获取到了消息，则通过dispatchMessage() 来分发给Handler处理消息。

6. 处理消息

消息最后到达了Handler的dispatchMessage() 方法。让我们看一下代码。

```
public void dispatchMessage(Message msg) {
        // 如果Message有自己的callback，就由Message的callback处理
        if (msg.callback != null) {
            handleCallback(msg);
        } else {
            //如果Handler有自己的mCallback，就由Handler的mCallback处理
            if (mCallback != null) {
                if (mCallback.handleMessage(msg)) {
                    return;
                }
            }
            //默认的处理消息的方法
            handleMessage(msg);
        }
    }
```

通过代码就可以知道消息处理的优先级。

1. 优先级最高的是message中自己的回调，这里的自己的回调是handler.post(runnable) 中的runnable（msg.callback = runnable ）。
2. 如果message中没有设置回调，则判断Handler中是否有自己的回调。这里的回调是我们在Handler中重写的handlerMessage() 方法。
3. 最后是默认的内部的handleMessage() 方法。

源码的分析差不多了，下一个文章是Handler的具体的使用方法和示例。