# [Android Handler消息传递机制详解](http://www.cnblogs.com/xsyulinzi/p/4323662.html)

1.为什么要用Handler

　　出于性能优化的考虑,Android UI操作并不是线程安全,如果有多个线程并发操作UI组件,可能导致线程安全问题。可以设想下,如果在一个Activity中有多个线程去更新UI,并且都没有加锁机制,可能会导致什么问题? 界面混乱,如果加锁的话可以避免该问题但又会导致性能下降。因此,Android规定只允许UI线程修改Activity的UI组件。当程序第一次启动时,Android会同时启动一条主线程(Main Thread),主线程主要负责处理与UI相关的事件,比如用户按钮事件,并把相关的事件分发到对应的组件进行处理,因此主线程又称为UI线程。那么怎么在新启动的线程中更新UI组件呢,这就需要借助handler的消息传递机制来实现了。

2.Handler简介

　　Handler类的主要作用主要有两个:

　　　　1>在新启动的线程中发送消息

　　　　2>在主线程中获取和处理消息

　　Handler类包含如下方法用于发送、处理消息。(这里只列出常用的方法,如果想获取更多方法,建议查看api文档:http://developer.android.com/reference/android/os/Handler.html)

　　　　♦ void handlerMessage(Message msg):处理消息的方法,该方法通常用于被重写。

　　　　♦ final boolean hasMessage(int what):检查消息队列中是否包含what属性为指定值的消息。

　　　　♦ sendEmptyMessage(int what):发送空消息

　　　　♦ final boolean sendMessage(Message msg):立即发送消息,注意这个返回值,如果message成功的被放到message queue里面则返回true,反之,返回false;(个人建议:对于这类问题不必主观去记它,当实际使用时,直接查看源码即可,源码中有详细的注释)

3.Handler、Message、Looper、MessageQueue之间的关系、工作原理

　　为了更好的理解Handler,先来看看和Handler相关的一些组件:

　　　　Message:Handler发送、接收和处理的消息对象

　　　　Looper:每个线程只能拥有一个Looper.它的looper()方法负责循环读取MessageQueue中的消息并将读取到的消息交给发送该消息的handler进行处理。

　　　　MessageQueue:消息队列,它采用先进先出的方式来管理Message。程序在创建Looper对象时,会在它的构造器中创建MessageQueue。源码如下:

1 　　private Looper(boolean quitAllowed) {

2 mQueue = new MessageQueue(quitAllowed);

3 mThread = Thread.currentThread();

4 }

　　　　从源码第2行中可以看出,在创建Looper对象时会创建一个与之关联的MessageQueue对象。构造器是private修饰的,所以程序员是无法创建Looper对象的。

　　　　Handler:前面说Handler作用有两个---发送消息和处理消息,Handler发送的消息必须被送到指定的MessageQueue,也就是说,要想Handler正常工作必须在当前线程中有一个MessageQueue,否则消息没法保存。而MessageQueue是由Looper负责管理的,因此要想Handler正常工作,必须在当前线程中有一个Looper对象,这里分为两种情况:

　　　　　　1>主线程(UI线程),系统已经初始化了一个Looper对象,因此程序直接创建Handler即可

　　　　　　2>程序员自己创建的子线程,这时,程序员必须创建一个Looper对象,并启动它。

　　　　创建Looper使用:Looper.prepare(),查看源码:

1 　　public static void prepare() {

2 prepare(true);

3 }

4

5　　　private static void prepare(boolean quitAllowed) {

6 if (sThreadLocal.get() != null) {

7 throw new RuntimeException("Only one Looper may be created per thread");

8 }

9 sThreadLocal.set(new Looper(quitAllowed));

10 }

11

12 　　private Looper(boolean quitAllowed) {

13 mQueue = new MessageQueue(quitAllowed);

14 mThread = Thread.currentThread();

15 }

　　　　通过方法调用,第9行创建Looper对象,创建Looper对象时同时会创建MessageQueue对象(第13行)。此外,可以看出prepare()允许一个线程最多有一个Looper被创建。

　　　　然后调用Looper的looper()方法来启动它,looper()使用一个死循环不断取出MessageQueue中的消息,并将消息发送给对应的Handler进行处理。下面是Looper类中looper()方法的部分源码:

1 for (;;) {

2 Message msg = queue.next(); // might block

3 if (msg == null) {

4 // No message indicates that the message queue is quitting.

5 return;

6 }

7

8 // This must be in a local variable, in case a UI event sets the logger

9 Printer logging = me.mLogging;

10 if (logging != null) {

11 logging.println(">>>>> Dispatching to " + msg.target + " " +

12 msg.callback + ": " + msg.what);

13 }

14

15 msg.target.dispatchMessage(msg);

16

17 if (logging != null) {

18 logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);

19 }

20

21 // Make sure that during the course of dispatching the

22 // identity of the thread wasn't corrupted.

23 final long newIdent = Binder.clearCallingIdentity();

24 if (ident != newIdent) {

25 Log.wtf(TAG, "Thread identity changed from 0x"

26 + Long.toHexString(ident) + " to 0x"

27 + Long.toHexString(newIdent) + " while dispatching to "

28 + msg.target.getClass().getName() + " "

29 + msg.callback + " what=" + msg.what);

30 }

31

32 msg.recycleUnchecked();

33 }

　　　　很明显第1行用了一个死循环,第2行从queue中取出Message,第15行通过dispatchMessage(Message msg)方法将消息发送给Handler。

4.HandlerThread介绍

　　Android API解释:

　　Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

　　意思是说:这个类启动一个新的线程并且创建一个Looper,这个Looper可以用来创建一个Handler类,完了之后一定要启动这个线程。

　　什么时候使用HandlerThread?

　　　　1.主线程需要通知子线程执行耗时操作(一般都是子线程执行耗时操作,完了之后,发送消息给主线程更新UI)。

　　　　2.开发中可能会多次创建匿名线程,这样可能会消耗更多的系统资源。而HandlerThread自带Looper使他可以通过消息来多次重复使用当前线程，节省开支；

　　下面是HandlerThread应用部分代码:

1 　　private static final String TAG = "MainActivity";

2 private static final int FLAG\_TEST = 1;

3

4 @Override

5 protected void onCreate(Bundle savedInstanceState) {

6 super.onCreate(savedInstanceState);

7 setContentView(R.layout.activity\_main);

8 Log.i(TAG,"main thread:"+Thread.currentThread());

9 HandlerThread thread = new HandlerThread("handler thread");

10 thread.start();//一定要启动该线程

11 Handler handler = new Handler(thread.getLooper()){

12 @Override

13 public void handleMessage(Message msg) {

14 Log.i(TAG,"handler thread:"+Thread.currentThread());

15 switch (msg.what){

16 case FLAG\_TEST:

17 //耗时操作...

18 break;

19 default:

20 break;

21 }

22 super.handleMessage(msg);

23 }

24 };

25 handler.sendEmptyMessage(FLAG\_TEST);

26 }

　　　　log:

com.example.administrator.handlertest I/MainActivity﹕ main thread:Thread[main,5,main]com.example.administrator.handlertest I/MainActivity﹕ handler thread:Thread[handler thread,5,main]

　　通过log可以看出handler处在一个子线程中,这样就能够执行一些耗时操作。

　　第十一行通过thread.getLooper()来创建handler,那么我们来看下getLooper()里面的源码:

1 public Looper getLooper() {

2 if (!isAlive()) {

3 return null;

4 }

5

6 // If the thread has been started, wait until the looper has been created.

7 synchronized (this) {

8 while (isAlive() && mLooper == null) {

9 try {

10 wait();

11 } catch (InterruptedException e) {

12 }

13 }

14 }

15 return mLooper;

16 }

　　　　看第8行代码,如果这个线程可用并且looper为null时,就会调用wait()方法,处于等待状态,这样可以有效的避免多线程并发操作引起的空指针异常。在thread启动时,会调用run()方法,再来看看run()方法里面的代码:

1 @Override

2 public void run() {

3 mTid = Process.myTid();

4 Looper.prepare();

5 synchronized (this) {

6 mLooper = Looper.myLooper();

7 notifyAll();

8 }

9 Process.setThreadPriority(mPriority);

10 onLooperPrepared();

11 Looper.loop();

12 mTid = -1;

13 }

　　第4行创建了Looper对象,第6、7行获取当前线程Looper之后调用notifyAll()方法。这时调用getLooper()方法返回一个Looper对象。

　　上面有提到使用HandlerThread避免多线程并发操作引起的空指针异常,这里解释下为什么:如果onCreate方法第11行通过程序员自定义的一个新线程创建handler时,很可能出现这样一个结果:创建handler的代码已经执行了,而新线程却还没有Looper.prepare()(创建Looper对象,那么这样就会导致空指针异常)。

　　对代码稍做修改:

1 package com.example.administrator.handlertest;

2

3 import android.os.Bundle;

4 import android.os.Handler;

5 import android.os.Looper;

6 import android.os.Message;

7 import android.support.v7.app.ActionBarActivity;

8 import android.util.Log;

9

10 public class MainActivity extends ActionBarActivity {

11

12 private static final String TAG = "MainActivity";

13 private static final int FLAG\_TEST = 1;

14

15 @Override

16 protected void onCreate(Bundle savedInstanceState) {

17 super.onCreate(savedInstanceState);

18 setContentView(R.layout.activity\_main);

19 Log.i(TAG,"main thread:"+Thread.currentThread());

20 // HandlerThread thread = new HandlerThread("handler thread");

21 // thread.start();//一定要启动该线程

22 MyThread thread = new MyThread();

23 thread.start();

24 Handler handler = new Handler(thread.looper){

25 @Override

26 public void handleMessage(Message msg) {

27 Log.i(TAG,"handler thread:"+Thread.currentThread());

28 switch (msg.what){

29 case FLAG\_TEST:

30 //耗时操作...

31 break;

32 default:

33 break;

34 }

35 super.handleMessage(msg);

36 }

37 };

38 handler.sendEmptyMessage(FLAG\_TEST);

39 }

40

41 static class MyThread extends Thread{

42 Looper looper;

43 @Override

44 public void run() {

45 Looper.prepare();looper = Looper.myLooper();

46 //...

47 Looper.loop();

48 }

49 }

50 }

　　　　运行结果:

Caused by: java.lang.NullPointerException

at android.os.Handler.<init>(Handler.java:234)

at android.os.Handler.<init>(Handler.java:142)

at com.example.administrator.handlertest.MainActivity$1.<init>(MainActivity.java:24)

at com.example.administrator.handlertest.MainActivity.onCreate(MainActivity.java:24)

at android.app.Activity.performCreate(Activity.java:5211)

at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1151)

at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2341)

　　　　从异常信息第4行中可以看出:onCreate()方法第24行thread.looper是一个null.这时因为还没等新线程创建Looper,Handler就已经创建了。如果在第23行thread.start()后面休眠几秒就不会报空指针异常了。

最后补充一点,Android判断当前更新UI的线程是否是主线程的对象ViewRootImpl对象在onResume()中,所以只要子线程在onResume()之前完成更新UI也是能够实现的。这里只是简单提一下,知道就行,不过不要这么做。