# 深入理解Handler、Looper、Messagequeue

1 简介

先对这几个类做一下简单介绍。 Handler： 线程间通信的方式，主要用来发送消息及处理消息。 Looper： 为线程运行消息循环的类，循环取出MessageQueue中的Message；消息派发，将取出的Message交付给相应的Handler。 MessageQueue： 存放通过Handler发过来的消息，遵循先进先出原则。 Message：消息，线程间通信通讯携带的数据。

例如后台线程在处理数据完毕后需要更新UI，则可发送一条包含更新信息的Message给UI线程

2 Looper

源码路径：frameworks/base/core/java/android/os/Looper.java Looper主要工作：

自身实例的创建，创建消息队列，保证一个线程中最多有一个Looper实例。消息循环，从消息队列中取出消息，进行派发。

Looper用于为线程运行消息循环的类，默认线程没有与它们相关联的消息循环；如果要想在子线程中进行消息循环，则需要在线程中调用prepare()，创建Looper对象。然后通过loop()方法来循环读取消息进行派发，直到循环结束。

程序中使用Looper的地方：

主线程（UI线程） UI线程中Looper已经都创建好了，不用我们去创建和循环。普通线程 普通线程中使用Looper需要我们自己去prepare()、loop()。 看一下普通线程中创建使用Looper的方式,代码如下:

class LooperThread extends Thread {

public Handler mHandler;

public void run() {

Looper.prepare();

mHandler = new Handler() {

public void handleMessage(Message msg) {

// process incoming messages here

}

};

Looper.loop();

}

}

这段代码是Looper源码注释中给的典型列子，主要步骤：

Looper 准备，（Looper实例创建）；创建发送消息、处理消息的Handler对象；Looper开始运行。

印象中在UI线程没有出现过Looper相关的东东，这是因为UI线程中会自动创建Looper对象并进行消息循环，我们不再需要调用Looper.prepare()和Looper.loop()，但是在子线程中如果想要创建使用Handelr则需要向如上所示。 我们通过源码看一下Looper实例创建的方法：

public static void prepare() {

prepare(true);

}

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));

}

Looper构造方法是私有的，只能通过prepare()进行创建Looper对象。prepare()会调用私有方法prepare(boolean quitAllowed)。 第6行 sThreadLocal为ThreadLocal类型变量，用来存储线程中的Looper对象。 prepare方法中首先判断sThreadLocal是否存储对象，如果存储了则抛出异常，这是因为在同一个线程中Loop.prepare()方法不能调用两次，也就是同一个线程中最多有一个Looper实例（当然也可以没有，如果子线程不需要创建Handler时）。 该异常应该许多朋友都遇见过，如在UI线程中调用Looper.prepare()，系统会替UI线程创建Looper实例，所以不需要再次调用prepare()。

接着看Looper的构造器：

private Looper(boolean quitAllowed) {

mQueue = new MessageQueue(quitAllowed);

mRun = true;

mThread = Thread.currentThread();

}

在构造器中，创建了一个MessageQueue消息队列；然后获取当前的线程，使Looper实例与线程绑定。 由prepare方法可知一个线程只会有一个Looper实例，所以一个Looper实例也只有一个MessageQueue实例。但这并不代表一个线程只能有一个MessageQueue实例，这是为什么呢？很简单，我们可以自己new 一个MessageQueue实例就可以了，但这个MessageQueue并不是该线程中Handelr对应的消息队列。

接着看Looper的消息循环：

public static void loop() {

final Looper me = myLooper();

if (me == null) {

throw new RuntimeException("No Looper; Looper.prepare() wasn't called on this thread.");

}

//通过Looper实例获取消息队列

final MessageQueue queue = me.mQueue;

// Make sure the identity of this thread is that of the local process,

// and keep track of what that identity token actually is.

Binder.clearCallingIdentity();

final long ident = Binder.clearCallingIdentity();

for (;;) { //消息循环

//从消息队列中取出一条消息，如果没有消息则会阻塞。

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

if (msg == null) {

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

return;

}

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

Printer logging = me.mLogging;

if (logging != null) {

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

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

}

//将消息派发给target属性对应的handler，调用其dispatchMessage进行处理。

msg.target.dispatchMessage(msg);

if (logging != null) {

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

}

// Make sure that during the course of dispatching the

// identity of the thread wasn't corrupted.

final long newIdent = Binder.clearCallingIdentity();

if (ident != newIdent) {

//log

}

msg.recycle();

}

}

loop()函数是静态的，所以它只能访问静态数据。 第2行myLooper()函数也是静态的，其代码如下

return sThreadLocal.get()

获取sThreadLocal存储的Looper实例，如果为空则抛出异常，这也说明loop()方法必须在prepare()方法之后才能调用。 第7行 通过Looper对象获取消息队列。 然后进行消息循环，从队列中获取消息，把消息交给msg的target的dispatchMessage方法进行处理，也就是交给handler进行处理，这个稍后说Handler时再细说。 然后调用msg.recycle(); 释放msg。

Looper.loop()是给死循环，那如何终止消息循环呢？我们可以调用Looper的quit方法或quitSafely方法。 quit方法作用是把MessageQueue消息池中所有的消息全部清空，无论是延迟消息还是非延迟消息。 quitSafely只会清空MessageQueue消息队列中所有的延迟消息，并将消息池中所有的非延迟消息派发出去让Handler去处理，quitSafely相比于quit方法安全之处在于清空消息之前会派发所有的非延迟消息。

总结：

UI线程会自动创建Looper实例、并且调用loop()方法，不需要我们再调用prepare()和loop().Looper与创建它的线程绑定，确保一个线程最多有一个Looper实例，同时一个Looper实例只有一个MessageQueue实例。loop()函数循环从MessageQueue中获取消息，并将消息交给消息的target的dispatchMessage去处理。如果MessageQueue中没有消息则获取消息可能会阻塞。通过调用Looper的quit或quitsafely终止消息循环。

3 Handler

源码路径：frameworks/base/core/java/android/os/Handler.java Handler主要职责：

发送消息给MessageQueue（消息队列）；处理Looper派送过来的消息； 我们使用Handler一般都要初始化一个Handler实例。看下Handler的构造函数：

public Handler() {

this(null, false);

}

public Handler(Callback callback, boolean async) {

//。。。。

mLooper = Looper.myLooper();

if (mLooper == null) {

throw new RuntimeException(

"Can't create handler inside thread that has not called Looper.prepare()");

}

mQueue = mLooper.mQueue;

mCallback = callback;

mAsynchronous = async;

}

第8行 Looper.myLooper();获取当前线程保存的Looper实例，如果当前线程没有Looper实例则会抛出异常。这也就是说在线程中应该先创建Looper实例（通过Looper.prepare()），然后才可以创建Handler实例。 第13行 获取Looper实例所保存的MessageQueue。之后使用Handler sendMesage、post都会将消息发送到该消息队列中。保证handler实例与该线程中唯一的Looper对象、及该Looper对象中的MessageQueue对象联系到一块。

3.1sendMessage

接着看一下平常使用Handler发送消息，先看sendMessage的流程：

public final boolean sendMessage(Message msg)

{

return sendMessageDelayed(msg, 0);

}

public final boolean sendMessageDelayed(Message msg, long delayMillis)

{

if (delayMillis < 0) {

delayMillis = 0;

}

return sendMessageAtTime(msg, SystemClock.uptimeMillis() + delayMillis);

}

public boolean sendMessageAtTime(Message msg, long uptimeMillis) {

MessageQueue queue = mQueue;

if (queue == null) {

RuntimeException e = new RuntimeException(

this + " sendMessageAtTime() called with no mQueue");

Log.w("Looper", e.getMessage(), e);

return false;

}

return enqueueMessage(queue, msg, uptimeMillis);

}

sendMessage最终调用到enqueueMessage函数，接着看下enqueueMessage。

private boolean enqueueMessage(MessageQueue queue, Message msg, long uptimeMillis) {

msg.target = this;

if (mAsynchronous) {

msg.setAsynchronous(true);

}

return queue.enqueueMessage(msg, uptimeMillis);

}

在enqueueMessage中，首先设置msg的target属性，值为this。之前在Looper的loop方法中，从消息队列中取出的msg，然后调用msg.target.dispatchMessage(msg);其实也就是调用当前handler的dispatchMessage函数。 然后调用queue的dispatchMessage方法，将Handler发出的消息，保存到消息队列中。

**3.2 pos**t

看一下post方法

public final boolean post(Runnable r)

{

return sendMessageDelayed(getPostMessage(r), 0);

}

private static Message getPostMessage(Runnable r) {

Message m = Message.obtain();

m.callback = r;

return m;

}

post方法中调用getPostMessage方法，创建一个Message对象，设置此Message对象的callback属性为创建Runnable对象。 然后调用sendMessageDelayed，最终和sendMessage一样，都是调用到sendMessageAtTime。调用enqueueMessage方法，将此msg添加到MessageQueue中。 也就是post(Runnable r) 并没有创建线程。其run方法是在Handler对应的线程中运行的。

3.3 dispatchMessage

这里主要说下handler是如何处理消息的。在Looper.loop方法中通过获取到的msg，然后调用msg.target.dispatchMessage(msg);也就是调用handler的dispatchMessage方法，看下Handler中dispatchMessage源码

public void dispatchMessage(Message msg) {

if (msg.callback != null) {

handleCallback(msg);

} else {

if (mCallback != null) {

if (mCallback.handleMessage(msg)) {

return;

}

}

handleMessage(msg);

}

}

在dispatchMessage方法中首先判断msg的callback属性，如果不为空则调用handleCallback函数， handleCallback函数如下：

private static void handleCallback(Message message) {

message.callback.run();

}

handleCallback函数中messag.callback也就是我们传的Runnable对象，也就是调用Runnable对象的run方法。 如果msg.callback属性为空，判断Handler属性mCallback是否为空， 不为空则让mCallback处理该msg。 mCallback为空则调用Handler的handleMessage，这就是我们创建Handler对象时一般都实现其handleMessage方法的原因。

4 MessageQueue

源码路径：frameworks/base/core/java/android/os/MessageQueue.java MessageQueue 消息队列：

enqueueMessage将消息加入队列next从队列取出消息removeMessage移除消息

MessageQueue内部是如何管理这些消息队列的就先不说了，之后又空再好好分析一下。

5 总结

本文分析了下Handler、Looper、MessageQueue之间的联系，及handler进程间通信的原理。了解到Handler不仅仅可以更新UI，也可以在一个子线程中去创建一个Handler，然后使用这个handler实例在任何其他线程中发送消息，最终处理这些消息的代码都会在你创建Handler实例的线程中运行。