

# Hongyang

生命不息，奋斗不止，万事起于忽微，量变引起质变

 目录视图 摘要视图 订阅

## 个人资料



鸿洋\_

+ 关注

✉ 发私信



访问：11978971次

积分：44709

等级：

排名：第58名

原创：194篇

转载：0篇

译文：6篇

评论：12991条

## 原 Android 异步消息处理机制 让你深入理解 Looper、Handler、Message三者关系

标签：[Android](#) [Looper](#) [Handler](#) [Message](#)

2014-08-07 09:17

 95220人阅读

 评论(102)

 收藏

 举报

分类：[【android 进阶之路】（70）](#) [【Android 源码解析】（28）](#)

版权声明：本文为博主原创文章，未经博主允许不得转载。

[目录\(?\)](#)

[\[+\]](#)

转载请标明出处：<http://blog.csdn.net/Imj623565791/article/details/38377229>，本文出自【张鸿洋的博客】


很多人面试肯定都被问到过，请问Android中的Looper，Handler，Message有什么关系？本篇博客目的首先为大家从源码角度开始分析，然后给出一个容易记忆的结论。

## 1、概述

Handler、Looper、Message 这三者都与Android异步消息处理线程相关的概念。那么什么叫异步消息处理线程呢？

异步消息处理线程启动后会进入一个无限的循环体之中，每循环一次，从其内部的消息队列中取出一个消息，然后回调相应的消息处理函数，执行完成一个消息后则继续循环。若消息队列为空，线程则会阻塞等待。

 快速回复

 我要收藏

## 我的微信公众号

[点击直达推送文章汇总](#)

长期为您推荐优秀博文、开源项目、视频等，进入还有好玩的等着你，欢迎扫一扫。



## 联系方式

[新动态](#)

[给我写信](#)

QQ群：

**497438697**

请勿重复加群，Thx

## 文章分类

- [【Android 5.x】](#) (11)
- [【Android 精彩案例】](#) (37)
- [【Android 源码解析】](#) (29)
- [【Android 自定义控件实战】](#) (29)
- [【Android 自定义控件之起步】](#) (7)
- [【Android 快速开发】](#) (12)
- [【Android 原生开发游戏】](#) (3)
- [【Java 并发专题】](#) (15)
- [【android 进阶之路】](#) (71)
- [【Java 设计模式】](#) (10)

说了这一堆，那么和Handler、Looper、Message有啥关系？其实Looper负责的就是创建一个MessageQueue，然后进入一个无限循环体不断从该MessageQueue中读取消息，而消息的创建者就是一个或多个Handler。

## 2、源码解析

### 1、Looper

对于Looper主要是prepare()和loop()两个方法。

首先看prepare()方法

```
[java] public static final void prepare() {
01.     if (sThreadLocal.get() != null) {
02.         throw new RuntimeException("Only one Looper may be created per thread");
03.     }
04.     sThreadLocal.set(new Looper(true));
05. }
06.
```

sThreadLocal是一个ThreadLocal对象，可以在一个线程中存储变量。可以看到，在第5行，将一个Looper的实例放入了ThreadLocal，并且2-4行判断了sThreadLocal是否为null，否则抛出异常。这也就说明了Looper.prepare()方法不能被调用两次，同时也保证了一个线程中只有一个Looper实例~相信有些哥们一定遇到这个错误。

下面看Looper的构造方法：

```
[java] private Looper(boolean quitAllowed) {
01.     mQueue = new MessageQueue(quitAllowed);
02.     mRun = true;
03.     mThread = Thread.currentThread();
04. }
05.
```

在构造方法中，创建了一个MessageQueue（消息队列）。

然后我们看loop()方法：

```
[java] public static void loop() {
01.
```

微信关注我的公众号



快速回复

我要收藏


- [【Android 百度地图】](#) (4)  
[【html5 css3精彩案例】](#) (14)  
[【Android github 控件】](#) (10)  
[【Android 基础】](#) (16)  
[【Javascript】](#) (9)  
[【rabbitMQ 用法】](#) (5)  
[【Android微知识点】](#) (4)

#### 友情链接

[郭霖的博客](#)  
[夏安明的博客](#)  
[任玉刚的博客](#)  
[元斌的博客](#)  
[敬佩的孔老师](#)  
[foruok的订阅号程序视界](#)  
[OpenCV大神shiter](#)  
[专为Android程序员的导航](#)  
[泡在网上的日子](#)

#### 博客专栏

 **HTML5 & CSS3 实战**  
文章：11篇  
阅读：166341


 **设计模式融入生活**  
文章：10篇  
阅读：97637

 **Android 精彩案例**  
文章：67篇  
阅读：4910745

```
02.     final Looper me = myLooper();
03.     if (me == null) {
04.         throw new RuntimeException("No Looper; Looper.prepare() wasn't called on this thread.");
05.     }
06.     final MessageQueue queue = me.mQueue;
07.
08.     // Make sure the identity of this thread is that of the local process,
09.     // and keep track of what that identity token actually is.
10.     Binder.clearCallingIdentity();
11.     final long ident = Binder.clearCallingIdentity();
12.
13.     for (;;) {
14.         Message msg = queue.next(); // might block
15.         if (msg == null) {
16.             // No message indicates that the message queue is quitting.
17.             return;
18.         }
19.
20.         // This must be in a local variable, in case a UI event sets the logger
21.         Printer logging = me.mLogging;
22.         if (logging != null) {
23.             logging.println(">>>> Dispatching to " + msg.target + " " +
24.                 msg.callback + ": " + msg.what);
25.         }
26.
27.         msg.target.dispatchMessage(msg);
28.
29.         if (logging != null) {
30.             logging.println("<<<< Finished to " + msg.target + " " + msg.callback);
31.         }
32.
33.         // Make sure that during the course of dispatching the
34.         // identity of the thread wasn't corrupted.
35.         final long newIdent = Binder.clearCallingIdentity();
36.         if (ident != newIdent) {
37.             Log.wtf(TAG, "Thread identity changed from 0x"
38.                 + Long.toHexString(ident) + " to 0x"
39.                 + Long.toHexString(newIdent) + " while dispatching to "
40.                 + msg.target.getClass().getName() + " "
41.                 + msg.callback + " what=" + msg.what);
42.         }
43.
44.         msg.recycle();
45.     }
46. }
```

微信关注我的公众号



 快速回复

 我要收藏

阅读排行

Android Https相关完全解析 ...	(1554166)
Android Fragment 真正的完...	(562934)
Android RecyclerView 使用...	(499898)
Android OkHttp完全解析 是...	(382637)
Android 自定义View (一)	(247007)
Android 属性动画 ( Property...	(232576)
Android Fragment 真正的完...	(210505)
Android 屏幕适配方案	(192323)
Android 手把手教您自定义Vi...	(177478)
Android 自定义RecyclerVie...	(155450)

文章搜索

Q

最新评论

- 2016一路有你, 2017一起同行  
nbcallum : 看大神的博客真的学习很多, 感谢, 希望新的一年能学习更多~
- Android 增量更新完全解析 是增量不是...  
叫我旺仔 : 以上用户言论只代表其个人观点, 不代表CSDN网站的观点或立场
- Android 自己实现 NavigationView [Des...  
下位子 : @qq\_19597141:这个我有办法, 看我的这篇文章 http://www.jianshu.com...
- Android 自己实现 NavigationView [Des...  
下位子 : @qq\_29269233:尽量不要用repla ce, 可以fragment生命周期重新走一遍, 尽量使用...
- Android 省市县 三级联动 ( android-wh...  
恰逢花开花香 : http://blog.csdn.net/u010 074743/article/details/54...
- Android 自定义View (一)  
qq\_34772386 : 学习了,谢谢楼主!!!

第2行：

```
public static Looper myLooper() {  
    return sThreadLocal.get();  
}
```

方法直接返回了sThreadLocal存储的Looper实例，如果me为null则抛出异常，也就是说looper方法必须在prepare方法

第6行：拿到该looper实例中的mQueue（消息队列）

13到45行：就进入了我们所说的无限循环。

14行：取出一条消息，如果没有消息则阻塞。

27行：使用调用 msg.target.dispatchMessage(msg);把消息交给msg的target的dispatchMessage方法去处理。Msg的target是什么呢？其实就是handler对象，下面会进行分析。

44行：释放消息占据的资源。

Looper主要作用：

- 1、与当前线程绑定，保证一个线程只会有一个Looper实例，同时一个Looper实例也只有一个MessageQueue。
- 2、loop()方法，不断从MessageQueue中去取消息，交给消息的target属性的dispatchMessage去处理。

好了，我们的异步消息处理线程已经有了消息队列（MessageQueue），也有了在无限循环体中取出消息的哥们，现在缺的就是发送消息的对象了，于是乎：Handler登场了。

2、Handler

使用Handler之前，我们都是初始化一个实例，比如用于更新UI线程，我们会在声明的时候直接初始化，或者在onCreate中初始化Handler实例。所以我们首先看Handler的构造方法，看其如何与MessageQueue联系上的，它在子线程中发送的消息（一般发送消息都在非UI线程）怎么发送到MessageQueue中的。

微信关注我的公众号



快速回复

我要收藏

[java] 复制 取消 分享

Android 自定义控件玩转字体变色 打造...

Dawish\_大D : mark

Android 一个改善的okHttp封装库

NoBug\_Android : 有人下载文件时遇到inProgress(float progress, long total, in...

Android 快速开发系列 ORMLite 框架最...

qq\_36216720 : java.lang.IllegalArgumentException: ORMLite does n...

Android 屏幕旋转 处理 AsyncTask 和 Pr...

冷暗雷 : @oWuGuanFengYue123:赞同

统计

微信公众号

```
01. public Handler() {
02.     this(null, false);
03. }
04. public Handler(Callback callback, boolean async) {
05.     if (FIND_POTENTIAL_LEAKS) {
06.         final Class<? extends Handler> klass = getClass();
07.         if ((klass.isAnonymousClass() || klass.isMemberClass() || klass.isLocalClass()) &&
08.             (klass.getModifiers() & Modifier.STATIC) == 0) {
09.             Log.w(TAG, "The following Handler class should be static or leaks might occur: " +
10.                 klass.getCanonicalName());
11.         }
12.     }
13.
14.     mLooper = Looper.myLooper();
15.     if (mLooper == null) {
16.         throw new RuntimeException(
17.             "Can't create handler inside thread that has not called Looper.prepare()");
18.     }
19.     mQueue = mLooper.mQueue;
20.     mCallback = callback;
21.     mAsynchronous = async;
22. }
```

微信关注我的公众号



14行：通过Looper.myLooper()获取了当前线程保存的Looper实例，然后在19行又获取了这个Looper实例中保存的MessageQueue（消息队列），这样就保证了handler的实例与我们Looper实例中MessageQueue关联上了。

然后看我们最常用的sendMessage方法

```
[java]
01. public final boolean sendMessage(Message msg)
02. {
03.     return sendMessageDelayed(msg, 0);
04. }
```

快速回复

☆ 我要收藏

```
[java]
01. public final boolean sendEmptyMessageDelayed(int what, long delayMillis) {
02.     Message msg = Message.obtain();
03.     msg.what = what;
04.     return sendMessageDelayed(msg, delayMillis);
}
```

```
05.     }
```

```
[java] 复制 清除 搜索
```

```
01. public final boolean sendMessageDelayed(Message msg, long delayMillis)
02. {
03.     if (delayMillis < 0) {
04.         delayMillis = 0;
05.     }
06.     return sendMessageAtTime(msg, SystemClock.uptimeMillis() + delayMillis);
07. }
```

微信关注我的公众号



```
[java] 复制 清除 搜索
```

```
01. public boolean sendMessageAtTime(Message msg, long uptimeMillis) {
02.     MessageQueue queue = mQueue;
03.     if (queue == null) {
04.         RuntimeException e = new RuntimeException(
05.             this + " sendMessageAtTime() called with no mQueue");
06.         Log.w("Looper", e.getMessage(), e);
07.         return false;
08.     }
09.     return enqueueMessage(queue, msg, uptimeMillis);
10. }
```

辗转反侧最后调用了sendMessageAtTime，在此方法内部有直接获取MessageQueue然后调用了enqueueMessage方法，我们将来看到此方法：

快速回复

☆ 我要收藏

```
[java] 复制 清除 搜索
```

```
01. private boolean enqueueMessage(MessageQueue queue, Message msg, long uptimeMillis) {
02.     msg.target = this;
03.     if (mAsynchronous) {
04.         msg.setAsynchronous(true);
05.     }
06.     return queue.enqueueMessage(msg, uptimeMillis);
07. }
```



enqueueMessage中首先为msg.target赋值为this，【如果大家还记得Looper的loop方法会取出每个msg然后交给msg.target.dispatchMessage(msg)去处理消息】，也就是把当前的handler作为msg的target属性。最终会调用queue的enqueueMessage的方法，也就是说handler发出的消息，最终会保存到消息队列中去。

现在已经很清楚了Looper会调用prepare()和loop()方法，在当前执行的线程中保存一个Looper实例，这个实例会保存一个Handler对象，然后当前线程进入一个无限循环中去，不断从MessageQueue中读取Handler发来的消息。然后再回调创建这个消息的dispatchMessage方法，下面我们赶快去看一看这个方法：

```
[java] 01. public void dispatchMessage(Message msg) {
      02.     if (msg.callback != null) {
      03.         handleCallback(msg);
      04.     } else {
      05.         if (mCallback != null) {
      06.             if (mCallback.handleMessage(msg)) {
      07.                 return;
      08.             }
      09.         }
      10.         handleMessage(msg);
      11.     }
      12. }
```

可以看到，第10行，调用了handleMessage方法，下面我们去看这个方法：

```
[java] 01. /**
      02.  * Subclasses must implement this to receive messages.
      03.  */
      04. public void handleMessage(Message msg) {
      05. }
      06.
```

微信关注我的公众号



快速回复

我要收藏

可以看到这是一个空方法，为什么呢，因为消息的最终回调是由我们控制的，我们在创建handler的时候都是复写handleMessage方法，然后根据msg.what进行消息处理。

例如：

```
[java] 01. private Handler mHandler = new Handler()
      02. {
      03.     public void handleMessage(android.os.Message msg)
      04.     {
      05.         switch (msg.what)
      06.         {
      07.             case value:
      08.
      09.                 break;
      10.
      11.             default:
      12.                 break;
      13.         }
      14.     };
      15. };
```

微信关注我的公众号



到此，这个流程已经解释完毕，让我们首先总结一下

1、首先Looper.prepare()在本线程中保存一个Looper实例，然后该实例中保存一个MessageQueue对象；因为Looper.prepare()在一个线程中只能调用一次，所以MessageQueue在一个线程中只会存在一个。

2、Looper.loop()会让当前线程进入一个无限循环，不端从MessageQueue的实例中读取消息，然后回调msg.target.dispatchMessage(msg)方法。

3、Handler的构造方法，会首先得到当前线程中保存的Looper实例，进而与Looper实例中的MessageQueue想关联。

4、Handler的sendMessage方法，会给msg的target赋值为handler自身，然后加入MessageQueue中。

5、在构造Handler实例时，我们会重写handleMessage方法，也就是msg.target.dispatchMessage(msg)最终调用的方法。

好了，总结完成，大家可能还会问，那么在Activity中，我们并没有显示的调用Looper.prepare()和Looper.loop()方法，为啥Handler可以成功创建呢，这是因为在Activity的启动代码中，已经在当前UI线程调用了Looper.prepare()和Looper.loop()方法。

### 3、Handler post



今天有人问我，你说Handler的post方法创建的线程和UI线程有什么关系？

其实这个问题也是出现这篇博客的原因之一；这里需要说明，有时候为了方便，我们会直接写如下代码：

```
[java] 01. mHandler.post(new Runnable()  
02.     {  
03.         @Override  
04.         public void run()  
05.         {  
06.             Log.e("TAG", Thread.currentThread().getName());  
07.             mTxt.setText("yoxi");  
08.         }  
09.     });
```

微信关注我的公众号



然后run方法中可以写更新UI的代码，其实这个Runnable并没有创建什么线程，而是发送了一条消息，下面看源码：

```
[java] 01. public final boolean post(Runnable r)  
02.     {  
03.         return sendMessageDelayed(getPostMessage(r), 0);  
04.     }
```

```
[java] 01. private static Message getPostMessage(Runnable r) {  
02.     Message m = Message.obtain();  
03.     m.callback = r;  
04.     return m;  
05. }
```

快速回复

我要收藏

可以看到，在getPostMessage中，得到了一个Message对象，然后将我们创建的Runnable对象作为callback属性，赋值给了此message。

注：产生一个Message对象，可以new，也可以使用Message.obtain()方法；两者都可以，但是更建议使用obtain方法，因为Message内部维护了一个Message池用于Message的复用，避免使用new重新分配内存。

```
[java] 01. public final boolean sendMessageDelayed(Message msg, long delayMillis)
```

```
02. {
03.     if (delayMillis < 0) {
04.         delayMillis = 0;
05.     }
06.     return sendMessageAtTime(msg, SystemClock.uptimeMillis() + delayMillis);
07. }
```

微信关注我的公众号



```
[java]
01. public boolean sendMessageAtTime(Message msg, long uptimeMillis) {
02.     MessageQueue queue = mQueue;
03.     if (queue == null) {
04.         RuntimeException e = new RuntimeException(
05.             this + " sendMessageAtTime() called with no mQueue");
06.         Log.w("Looper", e.getMessage(), e);
07.         return false;
08.     }
09.     return enqueueMessage(queue, msg, uptimeMillis);
10. }
```

最终和handler.sendMessage一样，调用了sendMessageAtTime，然后调用了enqueueMessage方法，给msg.target赋值为handler，最终加入MessageQueue。

可以看到，这里msg的callback和target都有值，那么会执行哪个呢？

其实上面已经贴过代码，就是dispatchMessage方法：

```
[java]
01. public void dispatchMessage(Message msg) {
02.     if (msg.callback != null) {
03.         handleCallback(msg);
04.     } else {
05.         if (mCallback != null) {
06.             if (mCallback.handleMessage(msg)) {
07.                 return;
08.             }
09.         }
10.         handleMessage(msg);
11.     }
12. }
```

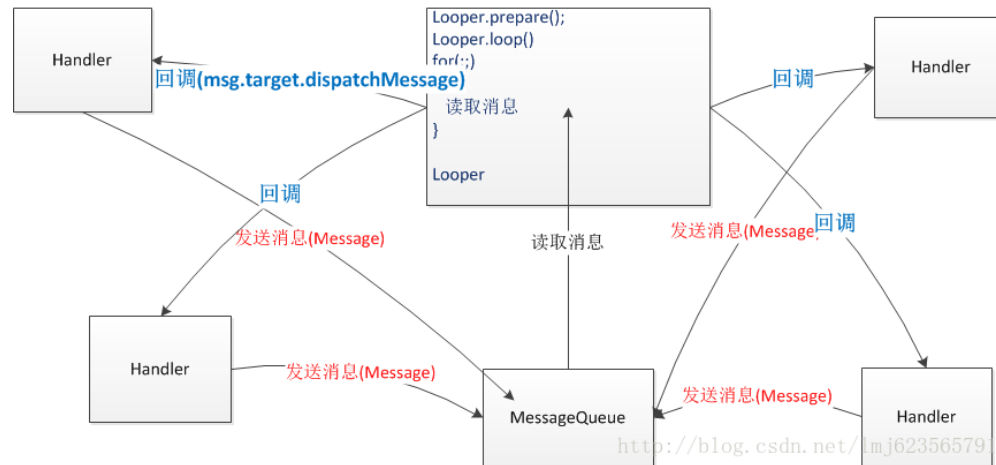
快速回复

☆ 我要收藏

第2行，如果不为null，则执行callback回调，也就是我们的Runnable对象。

好了，关于Looper，Handler，Message 这三者关系上面已经叙述的非常清楚了。

最后来张图解：



希望图片可以更好的帮助大家记忆~~

## 4、后话

其实Handler不仅可以更新UI，你完全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，最终处理消息的代码都会在你创建Handler实例的线程中运行。

[java]

```
01. new Thread()
02. {
03.     private Handler handler;
04.     public void run()
05.     {
06.
07.         Looper.prepare();
08.
09.         handler = new Handler()
10.         {
11.             public void handleMessage(android.os.Message msg)
12.             {
13.                 Log.e("TAG", Thread.currentThread().getName());
14.             };
```

微信关注我的公众号



```
15.         };\n        <pre code_snippet_id="445431" snippet_file_name="blog_20140808_19_1943618" name="code" class="java">
```

Loo

Android不仅给我们提供了异步消息处理机制让我们更好的完成UI的更新，其实也为我们提供了异步消息处理机制代码的原理，最好还可以将此设计用到其他的非Android项目中去~~

最新补充：

关于后记，有兄弟联系我说，到底可以在哪使用，见博客：[Android Handler 异步消息处理机制的妙用 创建强大的图片加载类](#)

微信关注我的公众号



顶

174

踩

10

- ▲ 上一篇 Android 自定义ViewGroup 实战篇 -> 实现FlowLayout
- ▼ 下一篇 Android Handler 异步消息处理机制的妙用 创建强大的图片加载类

快速回复

☆ 我要收藏

我的同类文章

【android 进阶之路】（70）

【Android 源码解析】（28）

- |   |  |
|---|--|
| • 2016一路有你，2017一起同行 2017-01-01 阅读 7047            | • Android 反编译初探 应用是如何被注... 2016-12-05 阅读 38554         |
| • Android Webp 完全解析 快来缩小ap... 2016-11-21 阅读 13239 | • 冰冻三尺非一日之寒-自学篇 浅谈个人... 2016-10-24 阅读 16836            |
| • Android 增量更新完全解析 是增量不... 2016-10-11 阅读 20459    | • Android 从StackTraceElement反观Lo... 2016-09-12 阅读 9529 |
| • Android 如何编写基于编译时注解的项目 2016-07-20 阅读 15150      | • Retrofit2 完全解析 探索与okhttp之间... 2016-05-04 阅读 50747    |