**handler机制**

参考：http://blog.csdn.net/lmj623565791/article/details/38377229/

1. Handler

Handler.sendMessage(Message)发送消息

Handler.handleMessage(Message)处理消息

1. Looper：轮询器，通过Looper.loop()从MmessageQueue中取消息。
2. MessageQueue：消息队列
3. Message：消息实体，可以通过new Message()或者Message.obtain()，Message.obtain()可以维护一个消息池。
4. 每个线程只有一个Looper，每个Looper都对应一个MessageQueue。
5. 主线程中的Handler
6. ActivityThread类中通过Looper.prepareMainLooper()方法创建Looper和MessageQueue

注：ActivityThread类的使用参考http://blog.csdn.net/luoshengyang/article/details/6747696

public static void prepareMainLooper() {

prepare(false);

synchronized (Looper.class) {

if (sMainLooper != null) {

throw new IllegalStateException("The main Looper has already been prepared.");

}

sMainLooper = myLooper();

}

}

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));

}

private Looper(boolean quitAllowed) {

mQueue = new MessageQueue(quitAllowed);

mThread = Thread.currentThread();

}

1. Handler通过调用Looper.myLooper()获取当前对象的Looper对象，从而与Looper建立联系

public Handler(Callback callback, boolean async) {

if (FIND\_POTENTIAL\_LEAKS) {

final Class<? extends Handler> klass = getClass();

if ((klass.isAnonymousClass() || klass.isMemberClass() || klass.isLocalClass()) &&

(klass.getModifiers() & Modifier.STATIC) == 0) {

Log.w(TAG, "The following Handler class should be static or leaks might occur: " +

klass.getCanonicalName());

}

}

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;

}

1. Looper.prepare()方法中创建Looper并将其保存在ThreadLocal变量中，Looper.prepare()在每个线程中只允许调用一次，否则会报异常。
2. 调用Looper.loop()方法，从MessageQueue队列中取出消息并通过Handler.dispatchMessage调用Handler.handleMessage()处理消息

public Handler() {

this(null, false);

}

public static void loop() {

final Looper me = myLooper();

if (me == null) {

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

}

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 (;;) {

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);

}

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

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

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

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

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

}

msg.recycleUnchecked();

}

}

1. Handler.sendMessage()将消息发送到MessageQueue中，并调用enqueueMessage方法将msg.target赋为当前Handler

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

msg.target = this;

if (mAsynchronous) {

msg.setAsynchronous(true);

}

return queue.enqueueMessage(msg, uptimeMillis);

}

1. 子线程中的Handler（与主线程中的Handler类似，但是需要手动调用Looper.prepare()和Looper.loop()方法）

new Thread(new Runnable() {

@Override

public void run() {

String msg;

Looper.prepare();

childHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

super.handleMessage(msg);

System.out.println("此消息来自-->>" + msg.obj+ "，在" + "btn的子线程当执行的")

}

};

Looper.loop();//开始轮循

}

}).start();