# **[Android消息传递之Handler消息机制](http://www.cnblogs.com/whoislcj/p/5590615.html)**

#### **Handler机制产生背景**

      一个Android应用程序被创建的时候都会创建一个UI主线程，但是有时我们会有一些比较耗时的操作，为了防止阻塞UI主线程，我们会将耗时的操作放到子线程中进行处理，处理完之后操作UI，但是Android不允许子线程操作UI，违背了Android单线程模型的原则（即 Android UI操作并不是线程安全的并且这些操作必须在UI线程中执行），所以Android通过Handler消息机制来实现线程之间的通讯。

#### **Handler机制主要角色**

    Message：消息，其中包含了消息ID，消息处理对象以及处理的数据等，由MessageQueue统一列队，终由Handler处理。

    Handler：处理者，负责Message的发送及处理。使用Handler时，需要实现handleMessage(Message msg)方法来对特定的Message进行处理，例如更新UI等。

    MessageQueue：消息队列，用来存放Handler发送过来的消息，并按照FIFO规则执行。当然，存放Message并非实际意义的保存，而是将Message以链表的方式串联起来的，等待Looper的抽取。

    Looper：消息泵，不断地从MessageQueue中抽取Message执行。因此，一个MessageQueue需要一个Looper。

    Thread：线程，负责调度整个消息循环，即消息循环的执行场所。

#### **Handler机制主要运用**

##### **1.）**发送消息，在不同的线程间发送消息，使用的方法为sendXXX();****

sendEmptyMessage(int);//发送一个空的消息

sendMessage(Message);//发送消息，消息中可以携带参数

sendMessageAtTime(Message, long);//未来某一时间点发送消息

sendMessageDelayed(Message, long);//延时Nms发送消息

 举例：

 主线程定义Handler

Handler mHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

super.handleMessage(msg);

switch (msg.what) {

case 0:

//完成主界面更新,拿到数据

String data = (String) msg.obj;

textView.setText(data);

break;

default:

break;

}

}

};

子线程发消息，通知Handler完成UI更新

private void getDataFromNet() {

new Thread(new Runnable() {

@Override

public void run() {

//耗时操作，完成之后发送消息给Handler，完成UI更新；

mHandler.sendEmptyMessage(0);

//需要数据传递，用下面方法；

Message msg = new Message();

msg.obj = "网络数据";//可以是基本类型，可以是对象，可以是List、map等； mHandler.sendMessage(msg);

}

}).start();

}

##### **2.）**计划任务，在未来执行某任务，使用的方法为postXXX();****

post(Runnable);//提交计划任务马上执行

postAtTime(Runnable, long);//提交计划任务在未来的时间点执行

postDelayed(Runnable, long);//提交计划任务延时Nms执行

示例：

主线程定义Handler

private Handler mHandler=new Handler();

子线程提交任务更新UI

private void getDataFromNet() {

new Thread(new Runnable() {

@Override

public void run() {

//耗时操作，完成之后提交任务更新UI

final String data = "网络数据";

mHandler.post(new Runnable() {

@Override

public void run() {

textView.setText(data);

}

});

}

}).start();

}

#### **Handler机制扩展**

    为了更加方便的使用Handler消息机制，Android也提供了几种扩展方式，内部实现都是基于Handler消息机制

##### **1.）**Activity.runOnUiThread(Runnable)****

private void getDataFromNet() {

new Thread(new Runnable() {

@Override

public void run() {

//耗时操作，完成之后提交任务更新UI

final String data = "网络数据";

runOnUiThread(new Runnable() {

@Override

public void run() {

textView.setText(data);

}

});

}

}).start();

}

##### ****2 .）View.post(Runnable)****

private void getDataFromNet() {

new Thread(new Runnable() {

@Override

public void run() {

//耗时操作，完成之后提交任务更新UI

final String data = "网络数据";

textView.post(new Runnable() {

@Override

public void run() {

textView.setText(data);

}

});

}

}).start();

}

5.)使用**AsyncTask代替Thread**

private void getDataFromNet() {

MyTask task = new MyTask();

task.execute();

}

private class MyTask extends AsyncTask {

//后台线程执行时

@Override

protected Object doInBackground(Object... params) {

////耗时操作，

String data = "网络数据";

return data;

}

//后台线程执行结束后的操作，其中参数result为doInBackground返回的结果

@Override

protected void onPostExecute(Object result) {

super.onPostExecute(result);

textView.setText((String) result);

}

}