题目：浅析 Android 线程模型

1 引言

Android 一词本义指机器人，Google 于 2007 年 11 月发布了以 Android 命名的开源移动设备综合平台，包括其基于 Linux 的操作系统、中间件和关键的手机应用。并且组建了开放手机联盟，其成员囊括了全球著名的各大手机生产 商和移动运营商。2008 年 8 月，Google 又发布了网上应用商店 Android Market。任何一个开发者只需要借助 Android 发 布的 SDK 开发手机应用，即可把开发的应用在 Android Market 上销售。目前 Android Market 上已经有一万多的应用程序，大大丰富了 Android 手机用户的功能。一个完整的产业链已经形成。因此开源 Android 吸引了原来越多的开发人员加入进来。本文将跟读者一起学习 Android 的线程模型。

2 Android 进程

在了解 Android 线程之间得先了解一下 Android 的进程。当一个程序第一次启动的时候，Android 会启动一个 LINUX 进程和一个主线程。默认的情况下，所有该程序的组件都将在该进程和线程中运行。同时，Android 会为每个应用程序分配一个单独的 LINUX 用户。Android 会劲量保留一个正在运行进程，只在内存资源出现不足时，Android 会参试停止一些进程从而释放足够的资源给其他新的进程使用， 也能保证用户正在访问的当前进程有足够的资源去及时的响应用户的事件。Android 会 根据进程中运行的组件类别以及组件的状态来判断该进程的重要性，Android 会 首先停止那些不重要的进程。按照重要性从高到低一共有五个级别：

l 前台进程

前台进程是用户当前正在使用的进程。只有一些前台进程可以在任何时候都存在。他们是最后一个被结束的，当内存低到根本连他们都不能运行的时候。一般来说， 在这种情况下，设备会进行内存调度，中止一些前台进程来保持对用户交互的响应。

l 可见进程

可见进程不包含前台的组件但是会在屏幕上显示一个可见的进程是的重要程度很高，除非前台进程需要获取它的资源，不然不会被中止。

l 服务进程

运行着一个通过 startService() 方法启动的 service，这个 service 不属于上面提到的 2 种更高重要性的。service 所在的进程虽然对用户不是直接可见的，但是他们执行了用户非常关注的任务（比如播放 mp3，从网络下载数据）。只要前台进程和可见进程有足够的内存，系统不会 回收他们。

l 后台进程

运行着一个对用户不可见的 activity（调用过 onStop() 方法).这些进程对用户体验没有直接的影响，可以在服务进程、可见进程、前台进 程需要内存的时候回收。通常，系统中会有很多不可见进程在运行，他们被保存在 LRU (least recently used) 列表中，以便内存不足的时候被第一时间回收。如果一个 activity 正 确的执行了它的生命周期，关闭这个进程对于用户体验没有太大的影响。

l 空进程

未运行任何程序组件。运行这些进程的唯一原因是作为一个缓存，缩短下次程序需要重新使用的启动时间。系统经常中止这些进程，这样可以调节程序缓存和系统缓 存的平衡。

Android 对进程的重要性评级的时候，选取它最高的级别。另外，当被另外的一个进程依赖的时候，某个进程的级别可能会增高。一个为其他进程服务的进程永远不会比被服 务的进程重要级低。因为服务进程比后台 activity 进程重 要级高，因此一个要进行耗时工作的 activity 最好启动一 个 service 来做这个工作，而不是开启一个子进程——特别 是这个操作需要的时间比 activity 存在的时间还要长的时 候。例如，在后台播放音乐，向网上上传摄像头拍到的图片，使用 service 可 以使进程最少获取到“服务进程”级别的重要级，而不用考虑 activity 目 前是什么状态。broadcast receivers 做费时的工作的时候，也应该启用一个服务而不是开一个线程。

2 单线程模型

当一个程序第一次启动时，Android 会同时启动一个对应的 主线程（Main Thread），主线程主要负责处理与 UI 相关的事件，如：用户的按键事件，用户接触屏幕的事件以及屏幕绘图事 件，并把相关的事件分发到对应的组件进行处理。所以主线程通常又被叫做 UI 线 程。在开发 Android 应用时必须遵守单线程模型的原则： 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。

2.1 案例演示

如果在没有理解这样的单线程模型的情况下，设计的程序可能会使程序性能低下，因为所有的动作都在同一个线程中触发。例如当主线程正在做一些比较耗时的操作的时候，如正从网络上下载一个大图片，或者访问数据库，由于主线程被这些耗时的操作阻塞住，无法及时的响应用户的事件，从用户的角度看会觉得程序已经死掉。如果程序长时间不响应，用户还可能得重启系统。为了避免这样的情况，Android 设 置了一个 5 秒 的超时时间，一旦用户的事件由于主线程阻塞而超过 5 秒 钟没有响应，Android 会 弹出一个应用程序没有响应的对话框。下面将通过一个案例来演示这种情况：

本程序将设计和实现查看指定城市的当天天气情况的功能，

1. 首先，需要选择一个天气查询的 服务接口，目前可供选择的接口很多，诸如 YAHOO 的 天气 API 和 Google 提 供的天气 API。 本文将选择 GOOGLE 的 天气查询 API。 该接口提供了多种查询方式，可以通过指定具体城市的经纬度进行查询，也可以通过城市名称进行查询。

2. 用户在输入框内输入需要查询的 城市名称，然后点击查询按钮

3. 当用户点击查询按钮后，使用已 经内置在 Android SDK 中的 HttpClient API 来调用 GOOGLE 的 天气查询 API， 然后解析返回的指定城市的天气信息，并把该天气信息显示在 Title 上

主要代码如下：

```
public class WeatherReport extends Activity implements OnClickListener {

    private static final String GOOGLE_API_URL = "http://www.google.com/ig/api?weather=";

    private static final String NETWORK_ERROR = "网络异常";

    private EditText editText;

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        editText = (EditText) findViewById(R.id.weather_city_edit);

        Button button = (Button) findViewById(R.id.goQuery);

        button.setOnClickListener(this);

    }

    @Override

    public void onClick(View v) {

        //获得用户输入的城市名称

        String city = editText.getText().toString();

        //调用 Google 天气 API 查询指定城市的当日天气 情况

        String weather = getWetherByCity(city);

        //把天气信息显示在 title 上
```

```
        setTitle(weather);
    }

    public String getWetherByCity(String city) {
        HttpClient httpClient = new DefaultHttpClient();
        HttpContext localContext = new BasicHttpContext();
        HttpGet httpGet = new HttpGet(GOOGLE_API_URL + city);
        try {
            HttpResponse response = httpClient.execute(httpGet, localContext);
            if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
                httpGet.abort();
            } else {
                HttpEntity httpEntity = response.getEntity();
                return parseWeather(httpEntity.getContent());
            }
        } catch (Exception e) {
            Log.e("WeatherReport", "Failed to get weather", e);
        } finally {
            httpClient.getConnectionManager().shutdown();
        }
        return NETWORK_ERROR;
    }
```

}

当用户输入城市名称，然后单击按钮进行查询后，程序会调用 Google API 的接口获得指定城市的当日天气情况。由于需要访问网络，所以当网络出现异常或者服务繁忙的时候都会使访问网络的动作很耗时。本文为了 要演示超时的现象，只需要制造一种网络异常的状况，最简单的方式就是断开网络连接，然后启动该程序，同时触发一个用户事件，比如按一下 MENU 键， 由于主线程因为网络异常而被长时间阻塞，所以用户的按键事件在 5 秒 钟内得不到响应，Android 会 提示一个程序无法响应的异常，如下图：

该对话框会询问用户 是继续等待还是强行退出程序。当你的程序需要去访问未知的网络的时候都会可能会发生类似的超时的情况，用户的响应得不到及时的回应会大大的降低用户体验。所以我们需要参试以别的方式来实现

2.1 子线程更新 UI

显然如果你的程序需要执行耗时的操作的话，如果像上例一样由主线程来负责执行 该操作是错误的。所以我们需要在 onClick 方 法中创建一个新的子线程来负责调用 GOOGLE API 来获得天气数据。刚接触 Android 的 开发者最容易想到的方式就是如下：

```
public void onClick(View v) {

    //创建一个子线程执行耗时的从网络上获取天气信息的操作

    new Thread() {

        @Override

        public void run() {

            //获得用户输入的城市名称

            String city = editText.getText().toString();

            //调用 Google 天气 API 查询指定城市的当日天气 情况

            String weather = getWetherByCity(city);

            //把天气信息显示在 title 上

            setTitle(weather);

        }
```

```
        }.start();

    }
```

但是很不幸，你会发 现 Android 会 提示程序由于异常而终止。为什么在其他平台上看起来很简单的代码在 Android 上运行的时候依然会出错呢？如果你观察 LogCat 中打印的日志信息就会发现这样的错误日志：

android.view.ViewRoot$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

从错误信息不难看出 Android 禁 止其他子线程来更新由 UI thread 创建的试图。本例中显示天气信息的 title 实际是就是一个由 UI thread 所创建的 TextView，所以参试在一个子线程中去更改 TextView 的时候就出错了。这显示违背了单线程模型的原则：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

2.2 Message Queue

在单线程模型下，为 了解决类似的问题，Android 设 计了一个 Message Queue(消息队列)， 线程间可以通过该 Message Queue 并结合 Handler 和 Looper 组 件进行信息交换。下面将对它们进行分别介绍：

- Message Queue

Message Queue 是一个消息队列，用来存放通过 Handler 发 布的消息。消息队列通常附属于某一个创建它的线程，可以通过 Looper.myQueue()得 到当前线程的消息队列。Android 在 第一启动程序时会默认会为 UI thread 创建一个关联的消息队列，用来管理程序的一些上层组件，activities，broadcast receivers 等等。你可以在自己的子线程中创建 Handler 与 UI thread 通讯。

- Handler

通过 Handler 你 可以发布或者处理一个消息或者是一个 Runnable 的 实例。没个 Handler 都 会与唯一的一个线程以及该线程的消息队列管理。当你创建一个新的 Handler 时候，默认情况下，它将关联到创建它的这个线程和该线程的消息队列。也就是说，如果你通过 Handler 发 布消息的话，消息将只会发送到与它关联的这个消息队列，当然也只能处理该消息队列中的消息。

主要的方法有：

1) public final boolean sendMessage(Message msg)

把消息放入该 Handler 所 关联的消息队列，放置在所有当前时间前未被处理的消息后。

2) public void handleMessage(Message msg)

关联该消息队列的线 程将通过调用 Handler 的 handleMessage 方 法来接收和处理消息，通常需要子类化 Handler 来 实现 handleMessage。

l Looper

Looper 扮演着一个 Handler 和 消息队列之间通讯桥梁的角色。程序组件首先通过 Handler 把 消息传递给 Looper，Looper 把 消息放入队列。Looper 也 把消息队列里的消息广播给所有的 Handler，Handler 接 受到消息后调用 handleMessage 进 行处理。

1) 可以通过 Looper 类 的静态方法 Looper.myLooper 得 到当前线程的 Looper 实 例，如果当前线程未关联一个 Looper 实 例，该方法将返回空。

2) 可以通过静态方法 Looper. getMainLooper 方法得到主线程的 Looper 实 例

线程，消息队列，Handler，Looper 之 间的关系可以通过一个图来展示：

在了解了消息队列及 其相关组件的设计思想后，我们将把天气预报的案例通过消息队列来重新实现：

在了解了消息队列及其相关组件的设计思想后，我们将把天气预报的案例通过消息队列来重新实现：

```
private EditText editText;

    private Handler messageHandler;

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        editText = (EditText) findViewById(R.id.weather_city_edit);

        Button button = (Button) findViewById(R.id.goQuery);

        button.setOnClickListener(this);

        //得到当前线程 的 Looper 实例，由于 当前线程是 UI 线程也可以 通过 Looper.getMainLooper()得到
```

```
        Looper looper = Looper.myLooper();
        //此处甚至可以 不需要设置 Looper，因为 Handler 默认就使用当 前线程的 Looper
        messageHandler = new MessageHandler(looper);
    }

    @Override
    public void onClick(View v) {
        //创建一个子线 程去做耗时的网络连接工作
        new Thread() {
            @Override
            public void run() {
                //活动用户输入 的城市名称
                String city = editText.getText().toString();
                //调用 Google 天气 API 查询指定城 市的当日天气情况
                String weather = getWetherByCity(city);
                //创建一个 Message 对象，并把得 到的天气信息赋值给 Message 对象
                Message message = Message.obtain();
                message.obj = weather;
                //通过 Handler 发布携带有天 气情况的消息
                messageHandler.sendMessage(message);
            }
        }.start();
```

```
}

//子类化一个 Handler
class MessageHandler extends Handler {
    public MessageHandler(Looper looper) {
        super(looper);
    }
    @Override
    public void handleMessage(Message msg) {
        //处理收到的消  息，把天气信息显示在 title 上
        setTitle((String) msg.obj);
    }
}
```

通过消息队列改写过后的天气预报程序已经可以成功运行，因为 Handler 的 handleMessage 方法实 际是由关联有该消息队列的 UI thread 调用，而在 UI thread 中更新 title 并没有违背 Android 的单线程模型的原 则。

2.3 AsyncTask

虽然借助消息队列已经可以较为完美的实现了天气预报的功能，但是你还是不得不自己管理子线程，尤其当你的需要有一些复杂的逻辑以及需要频繁的更新 UI 的时候，这样的方式使得你的代码难以阅读和理解。

幸运的是 Android 另外提供了一个工具类：AsyncTask。它使得 UI thread 的使用变得异常简单。它使创建需要与用户界面交互的长时间运行的任务变得更简单，不需要借助线程和 Handler 即可实现。

1) 子类化 AsyncTask

2) 实现 AsyncTask 中定义的下面一个或几个方法

1. onPreExecute(), 该方法将在执行实际的后台操作前被 UI thread 调用。可以在该方法中做一些准备工作，如在界面上显示一个进度条。

2.doInBackground(Params...), 将在 onPreExecute 方法执行后马上执行，该方法运行在后台线程中。这里将主要负责执行那些很耗时的后台计算工作。可以调用 publishProgress 方法来更新实时的任务进度。该方法是抽象方法，子类必须实现。

3. onProgressUpdate(Progress...),在 publishProgress 方 法被调用后，UI thread 将调用这个方法从而在界面上展示任务的进展情况，例如通过一个进度条进行展示。

4. onPostExecute(Result), 在 doInBackground 执行完成后，onPostExecute 方法将被 UI thread 调用，后台的计算结果将通过该方法传递到 UI thread.

为了正确的使用 AsyncTask 类，以下是几条必须遵守的准则：

1) Task 的实例 必须在 UI thread 中创建

2) execute 方 法必须在 UI thread 中调用

3) 不要手动的调用 onPreExecute(), onPostExecute(Result)，doInBackground(Params...), onProgressUpdate(Progress...)这几个方法

4) 该 task 只能被执行一次，否则多次调用时将会出现异常

下面我们将通过 AsyncTask 并且严格遵守上面的 4 条准则来改写天气预报的例子：

```
    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        editText = (EditText) findViewById(R.id.weather_city_edit);

        Button button = (Button) findViewById(R.id.goQuery);

        button.setOnClickListener(this);

    }

    public void onClick(View v) {

        //获得用户输 入的城市名称
```

```
        String city = editText.getText().toString();
    //必须每次都 重新创建一个新的 task 实例进行 查询，否则将提示如下异常信息
    //the task has already been executed (a task can be executed only once)
        new GetWeatherTask().execute(city);
    }
    class GetWeatherTask extends AsyncTask<String, Integer, String> {
        @Override
        protected String doInBackground(String... params) {
            String city = params[0];
            //调用 Google 天气 API 查询指定 城市的当日天气情况
            return getWetherByCity(city);
        }
        protected void onPostExecute(String result) {
            //把 doInBackground 处理的结果 即天气信息显示在 title 上
            setTitle(result);
        }
    }
```

注意这行代 码：new GetWeatherTask().execute(city); 值得一提的是必须每次都重新创建一个新的 GetWeatherTask 来执行后台任务，否则 Android 会提示“a task can be executed only once”的错误信息。

经过改写后的 程序不仅显得非常的简洁，而且还减少了代码量，大大增强了可读性和可维护性。因为负责更新 UI 的 onPostExecute 方 法是由 UI thread 调用，所以没有违背单线程模型的原则。良好的 AsyncTask 设计大大降低了我们犯错误的几率。