新研科技

求知若渴 求真教育

第十六章. 安卓消息处理机制

`android`

学习目标：

- 创建和开启线程,线程休眠,中断线程
- 循环者Looper
- 消息处理类Handler的应用
- 消息类Message的应用
- 在子线程中更新UI界面的方法

课程内容：

多线程

在现实生活中,很多事情都是同时进行的,例如,大家可以一边看书,一边喝咖啡,而计算机可以一喧播放音乐,一边打印文档.对于这种可以同时进行的任务,就可以用线程来表标,每个纯种完成一个任务,并与其他线程同时执行,这种机制被称为多线程.
创建线程
1,通过Thread类的构造方法创建线程

```
    Public class MyThread extents Thread{
        Public void run(){

        }
    }
```

2,通过实现runnale接口

```
    Public class MyThread implements Runnable{
        public void run(){

        }
    }
```

常用方法

```
开启线程:start();
线程休眠:sleep(long time);
```

异步任务AsyncTask

Android的AsyncTask轻量级简单的异步处理。
首先明确Android之所以有Handler和AsyncTask，都是为了不阻塞主线程（UI线程），且UI的更新只能在主线程中完成，因此异步处理是不可避免的。
Android为了降低这个开发难度，提供了AsyncTask。AsyncTask就是一个封装过的后台任务类，顾名思义就是异步任务。
AsyncTask直接继承于Object类，位置为`android.os.AsyncTask`。要使用AsyncTask工作我们要提供三个泛型参数，并重载几个方法(至少重载一个)。
AsyncTask定义了三种泛型类型 Params，Progress和Result。

- Params 启动任务执行的输入参数，比如HTTP请求的URL。
- Progress 后台任务执行的百分比。
- Result 后台执行任务最终返回的结果，比如String。

使用过AsyncTask最少要重写以下这两个方法：

```
doInBackground(Params…)
//后台执行，比较耗时的操作都可以放在这里。注意这里不能直接操作UI。此方法在后台线程执行，完成任务的主要工作，通常需要较长的时间。在执行过程中可以调用publicProgress(Progress…)来更新任务的进度。

onPostExecute(Result)
//相当于Handler 处理UI的方式，在这里面可以使用在doInBackground 得到的结果处理操作UI。 此方法在主线程执行，任务执行的结果作为此方法的参数返回
```

有必要的话你还得重写以下这三个方法，但不是必须的：

```
onProgressUpdate(Progress…)
//可以使用进度条增加用户体验度。此方法在主线程执行，用于显示任务执行的进度。

onPreExecute()
//这里是最终用户调用Excute时的接口，当任务执行之前开始调用此方法，可以在这里显示进度对话框。

onCancelled()
//用户调用取消时，要做的操作
```

使用AsyncTask类，以下是几条必须遵守的准则：

- Task的实例必须在UI thread中创建；
- execute方法必须在UI thread中调用；
- 不要手动的调用`onPreExecute()`，`onPostExecute(Result)`，`doInBackground(Params...)`，`onProgressUpdate(Progress...)`这几个方法；

> 注意：AsyncTask只能被执行一次，否则多次调用时将会出现异常；

Handler消息传递机制

android的消息处理有三个核心类：Looper,Handler和Message。其实还有一个Message Queue（消息队列），但是MQ被封装到Looper里面了，我们不会直接与Message Queue打交道，因此我没将其作为核心类。下面一一介绍：

线程的魔法师 Looper

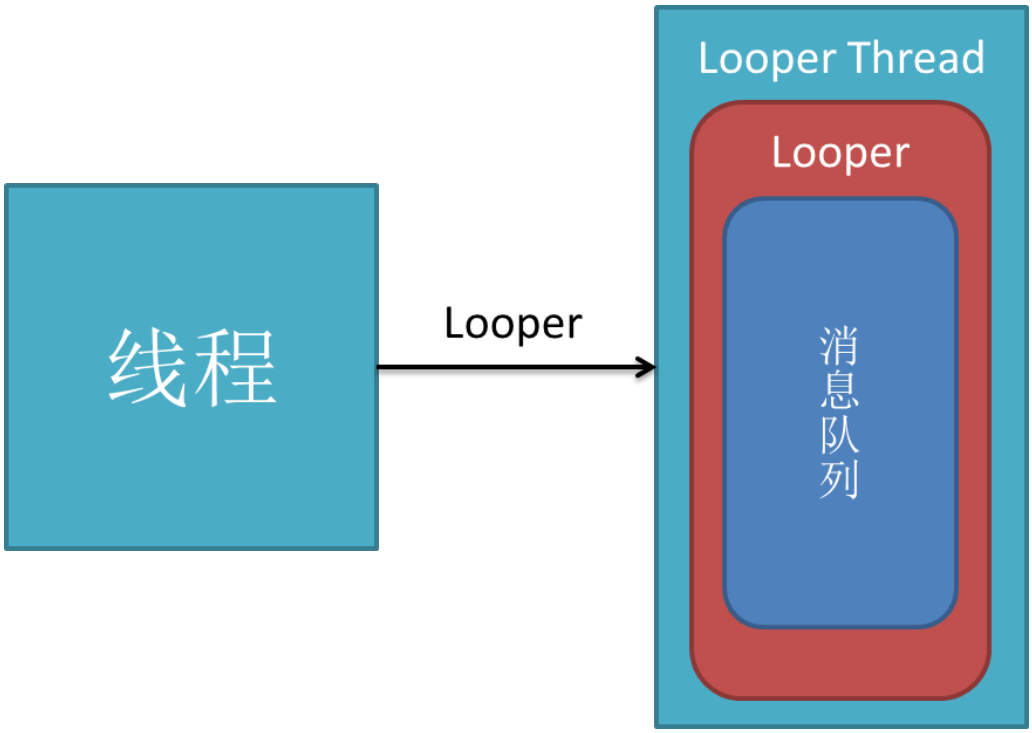
Looper的字面意思是“循环者”，它被设计用来使一个普通线程变成Looper线程。所谓Looper线程就是循环工作的线程。在程序开发中（尤其是GUI开发中），我们经常会需要一个线程不断循环，一旦有新任务则执行，执行完继续等待下一个任务，这就是Looper线程。使用Looper类创建Looper线程很简单：

```
1.      public class LooperThread extends Thread {
2.          @Override
3.          public void run() {
4.          // 将当前线程初始化为Looper线程
5.          Looper.prepare();
6.
7.          // ...其他处理，如实例化handler
8.
9.          // 开始循环处理消息队列
10.         Looper.loop();
11.         }
12.     }
```

通过上面两行核心代码，你的线程就升级为Looper线程了！

> 注意：`Looper.loop()`之后的方法不会再执行到

1)Looper.prepare()

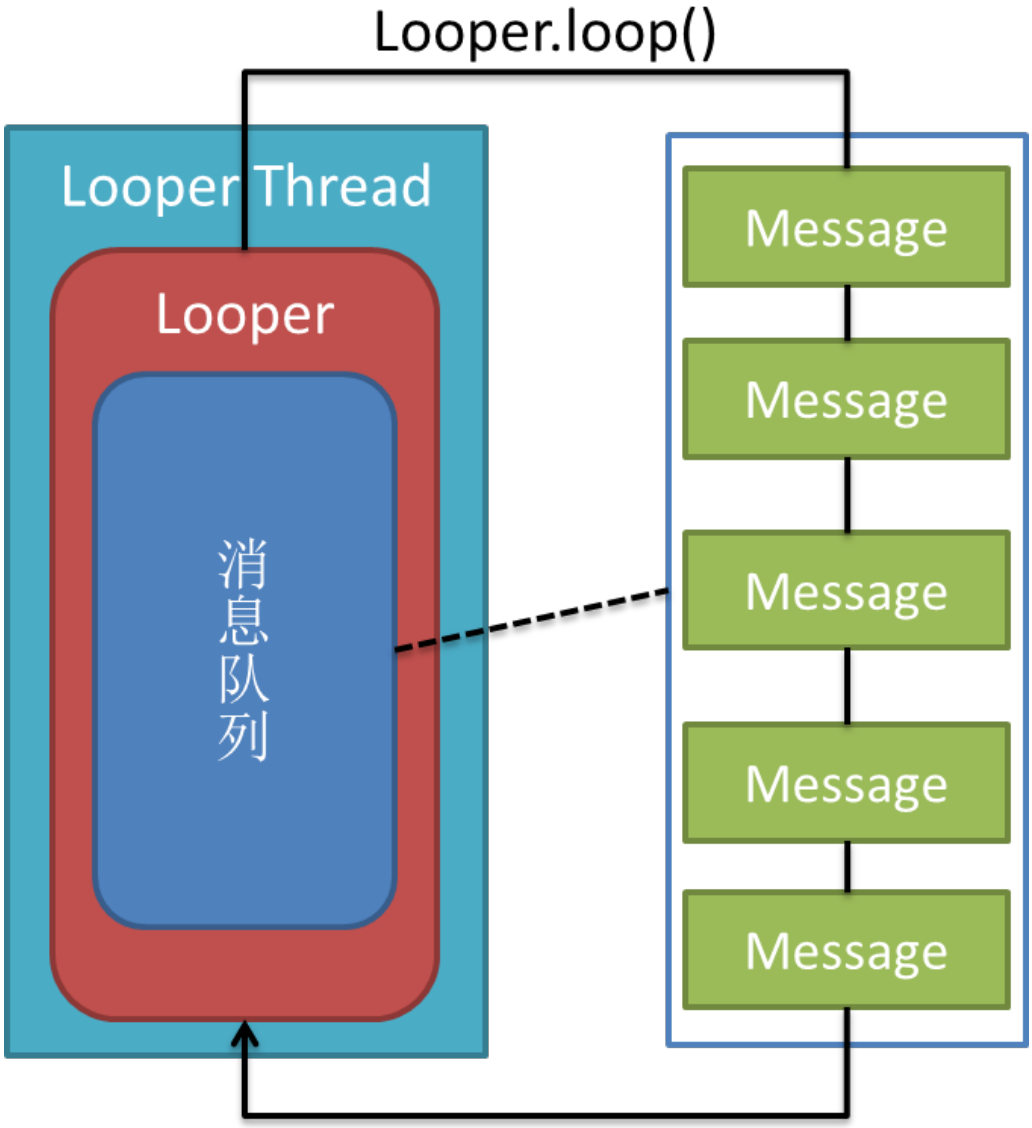


通过上图可以看到，现在你的线程中有一个Looper对象，它的内部维护了一个消息队列MQ。

注意，**一个Thread只能有一个Looper对象**

```
public static final void prepare() {
    if (sThreadLocal.get() != null) {
        // 试图在有Looper的线程中再次创建Looper将抛出异常
        throw new RuntimeException("Only one Looper may be created per thread");
    }
    sThreadLocal.set(new Looper());
    }
}
```

2）Looper.loop()



调用loop方法后，Looper线程就开始真正工作了，它不断从自己的MQ中取出队头的消息(也叫任务)执行。

除了prepare()和loop()方法，Looper类还提供了一些有用的方法，比如Looper.myLooper()得到当前线程looper对象：

```
1.      public static final Looper myLooper()
```

getThread()得到looper对象所属线程：

```
1.      public Thread getThread()
```

quit()方法结束looper循环：

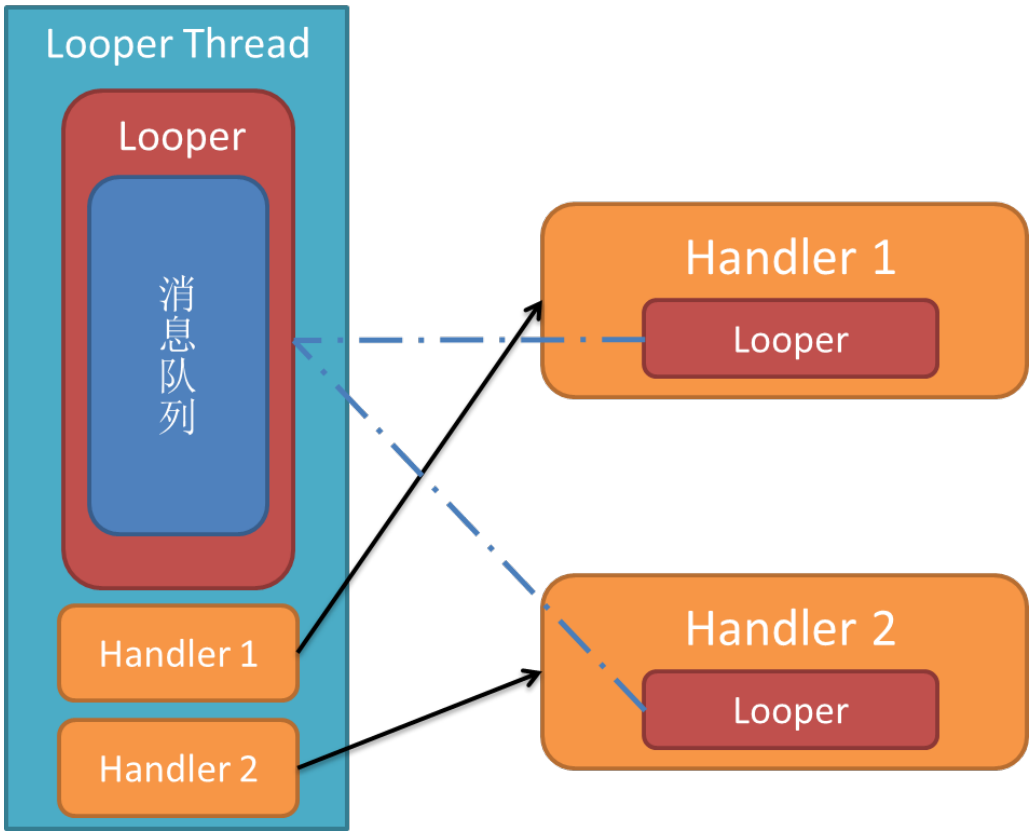
```
1.      public void quit()
```

总结

- 每个线程有且最多只能有一个Looper对象，它是一个ThreadLocal
- Looper内部有一个消息队列，loop()方法调用后线程开始不断从队列中取出消息执行
- Looper使一个线程变成Looper线程。

异步处理大师 Handler

什么是Handler？Handler扮演了往MQ上添加消息和处理消息的角色（只处理由自己发出的消息），即通知MQ它要执行一个任务(sendMessage)，并在loop到自己的时候执行该任务(handleMessage)，整个过程是异步的。handler创建时会关联一个looper，默认的构造方法将关联当前线程的looper，不过这也是可以set的。



可以看到，一个线程可以有多个Handler，但是只能有一个Looper！

Handler发送消息
有了handler之后，我们就可以使

用`post(Runnable)`,`postAtTime(Runnable,long)`,`postDelayed(Runnable, long)`,`sendEmptyMessage(int)`,`sendMessage(Message)`,`sendMessageAtTime(Message, long)`和`sendMessageDelayed(Message,long)`这些方法向MQ上发送消息了。光看这些API你可能会觉得handler能发两种消息，一种是Runnable对象，一种是message对象，这是直观的理解，但其实post发出的Runnable对象最后都被封装成message对象了

Message类

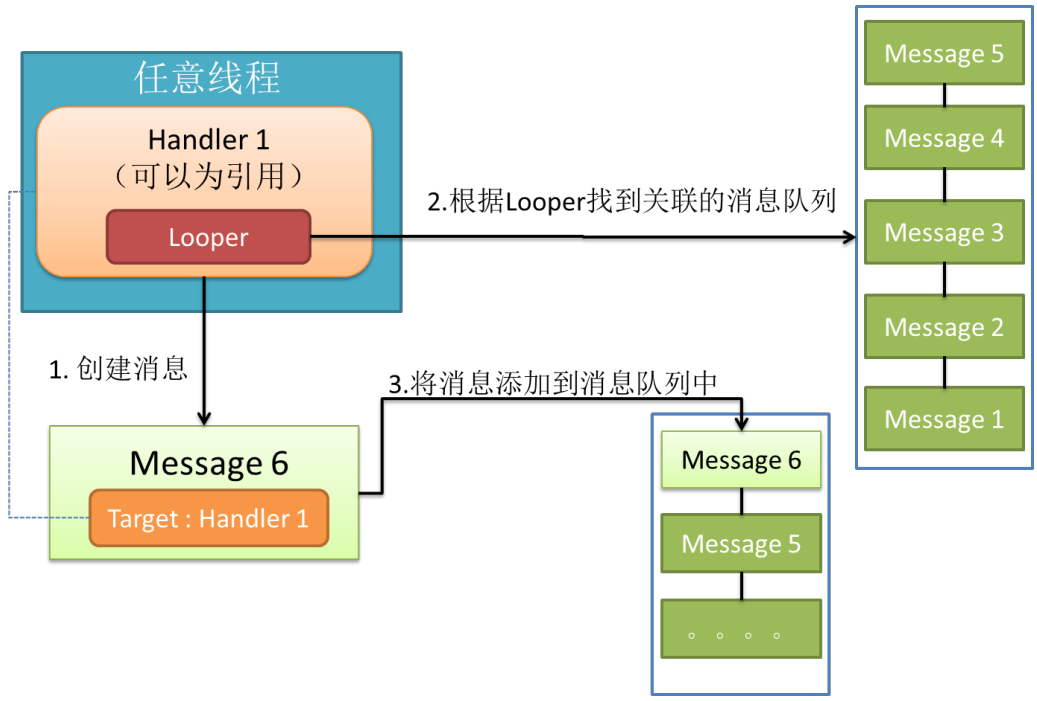
Message是线程之间传递信息的载体，包含了对消息的描述和任意的数据对象。Message被存放在MessageQueue中，一个MessageQueue中可以包含多个Message对象,Message中包含了两个额外的 int字段和一个object字段，这样在大部分情况下，使用者就不需要再做内存分配工作了。虽然Message的构造函数是public的，但是最好是使用`Message.obtain()`或`Handler.obtainMessage()`函数来获取Message对象，因为Message的实现中包含了回收再利用的机制，可以提供效率。

```
1.  arg1        int         用来存放整型数据
2.  arg2        int         用来存放整型数据
3.  obj         Object      用来存放发送给接收器的Object类型的任意对象
4.  replayTo    Messenger   用来指定此Message发送到何处的可选Messager对象
5.  what        int         用于指定用户自定义的消息代码，这样接收者可以了解这个消息
    的信息
```

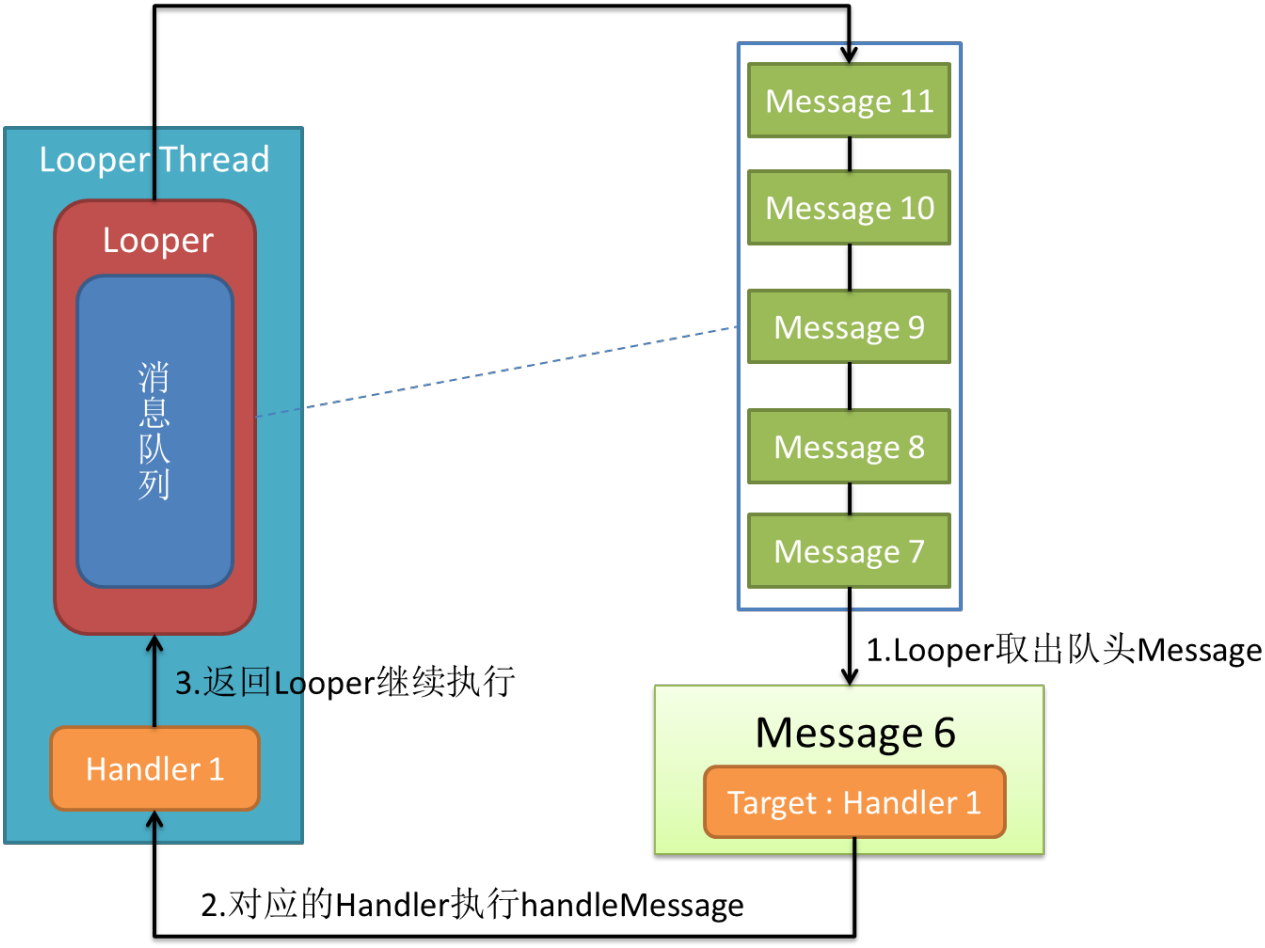
> 注意：使用Message类的属性可以携带int类型数据，如果要携带其它类型的数据，可以先将要携带的数据保存到Bundle对象中。然后通过Message类的setDate()方法将其添加到Message中。
>
> 如果一个Message只需要携带简单的int型信息，应优先使用`Message.arg1`和`Message.arg2`属性来传递信息，这比用Bundle更省内存。
>
> 尽可能使用`Message.what`来标识信息，以便用不同方式处理Message

Handler的用处

- handler可以在任意线程发送消息，这些消息会被添加到关联的MQ上。

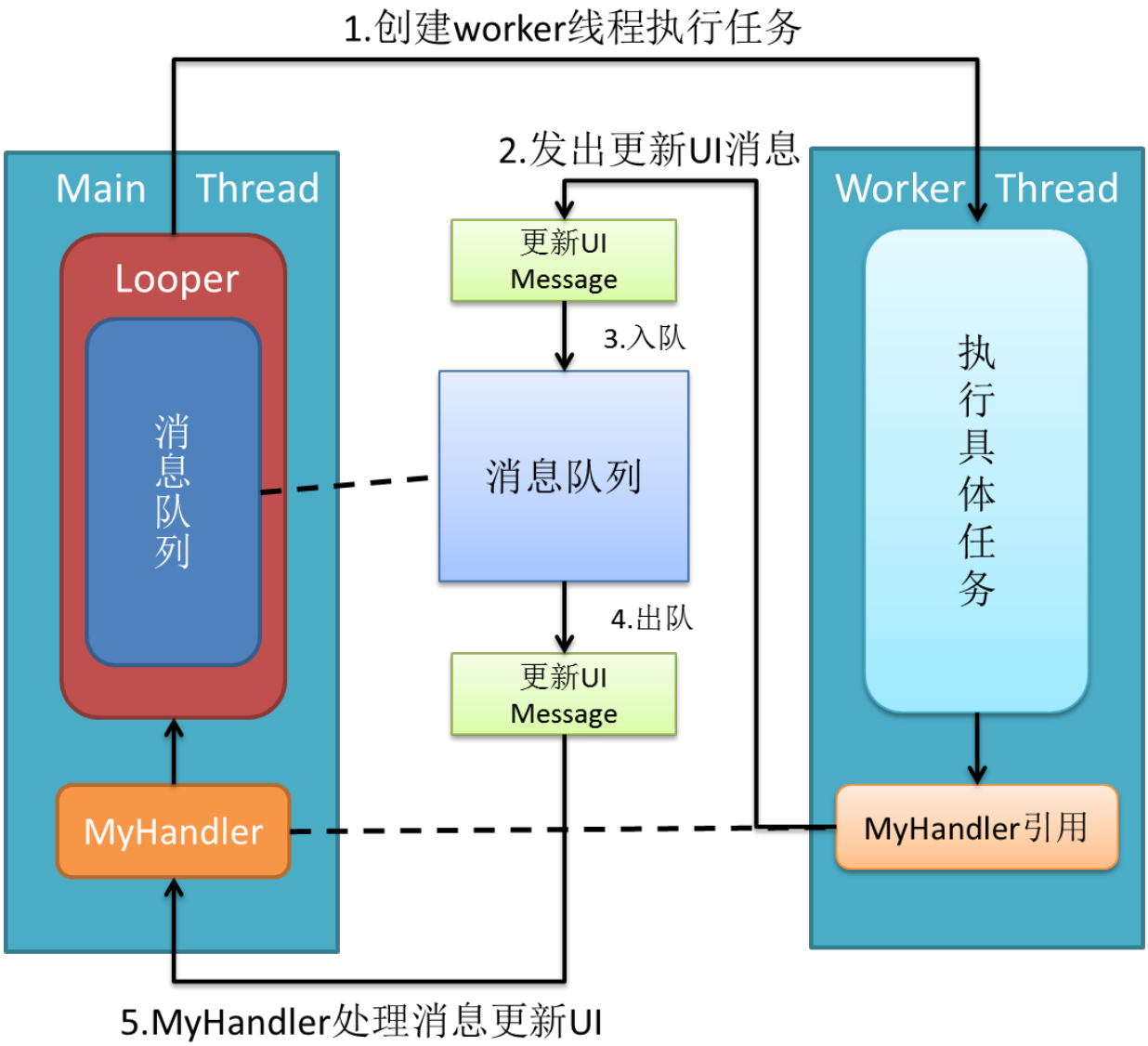


- handler是在它关联的looper线程中处理消息的。



就解决了android最经典的不能在其他非主线程中更新UI的问题。android的主线程也是一个looper线程(looper 在android中运用很广)，我们在其中创建的handler默认将关联主线程

MQ。因此，利用handler的一个solution就是在 activity中创建handler并将其引用传递给worker thread，worker thread执行完任务后使用handler发送消息通知activity更新UI。



学员作业

使用异步任务加载网络图片，显示在GridView中，在加载过程中，GridView显示等待加载动画，加载完毕后立即刷新UI

提示，加载网络图片可以使用BItmapFactory类，实例代码如下

```
        try {
            URL url = new URL("图片网址，必须携带http://");
            //打开流
            InputStream is = url.openStream();
```

```
            //将流生成位图
            Bitmap bmp = BitmapFactory.decodeStream(is);
            //关闭流
            is.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
```