**Handler+Runnable模式**

我们先看一个并不是异步线程加载的例子，使用 Handler+Runnable模式。

这里为何不是新开线程的原因请参看这篇文章：Android Runnable 运行在那个线程 这里的代码其实是在UI 主线程中下载图片的，而不是新开线程。

我们运行下面代码时，会发现他其实是阻塞了整个界面的显示，需要所有图片都加载完成后，才能显示界面。

见线程池的4个例子-例1.java

**Handler+Thread+Message模式**

这种模式使用了线程，所以可以看到异步加载的效果。

见线程池的4个例子-例2.java

**Handler+ExecutorService(线程池)+MessageQueue模式**

能开线程的个数毕竟是有限的，我们总不能开很多线程，对于手机更是如此。

这个例子是使用线程池。Android拥有与Java相同的ExecutorService实现，我们就来用它。

线程池的基本思想还是一种对象池的思想，开辟一块内存空间，里面存放了众多(未死亡)的线程，池中线程执行调度由池管理器来处理。当有线程任务时，从池中取一个，执行完成后线程对象归池，这样可以避免反复创建线程对象所带来的性能开销，节省了系统的资源。

见线程池的4个例子-例3.java

**Handler+ExecutorService(线程池)+MessageQueue+缓存模式**

下面比起前一个做了几个改造:

•把整个代码封装在一个类中

•为了避免出现同时多次下载同一幅图的问题,使用了本地缓存

见线程池的4个例子-例4.java

说明：

final参数是指当函数参数为final类型时，你可以读取使用该参数，但是无法改变该参数的值。参看：Java关键字final、static使用总结

这里使用SoftReference 是为了解决内存不足的错误（OutOfMemoryError）的，更详细的可以参看：内存优化的两个类:SoftReference 和 WeakReference

前段调用:

线程池的4个例子-代码5.java