* **HandlerThread**

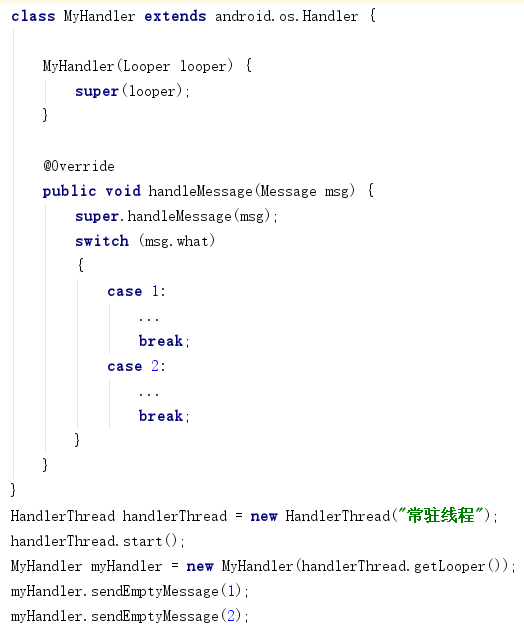
普通的子线程创建一个Thread类，然后实现它的run方法，当run方法执行完了这个线程也就结束了。子线程消息循环就是创建一个常驻线程来不定期执行耗时任务，创建一个常驻行线程很简单，不过Android系统已经为我们准备好了这个轮子HandlerThread，它继承Thread类，并实现了run方法：



首先我们看到的是，HandlerThread类继承了Thread类，因此，通过它可以在应用程序中创建一个子线程，其次我们看到在它的run函数中，会进入一个消息循环中，因此，这个子线程可以常驻在应用程序中，直到它接收收到一个退出消息为止。

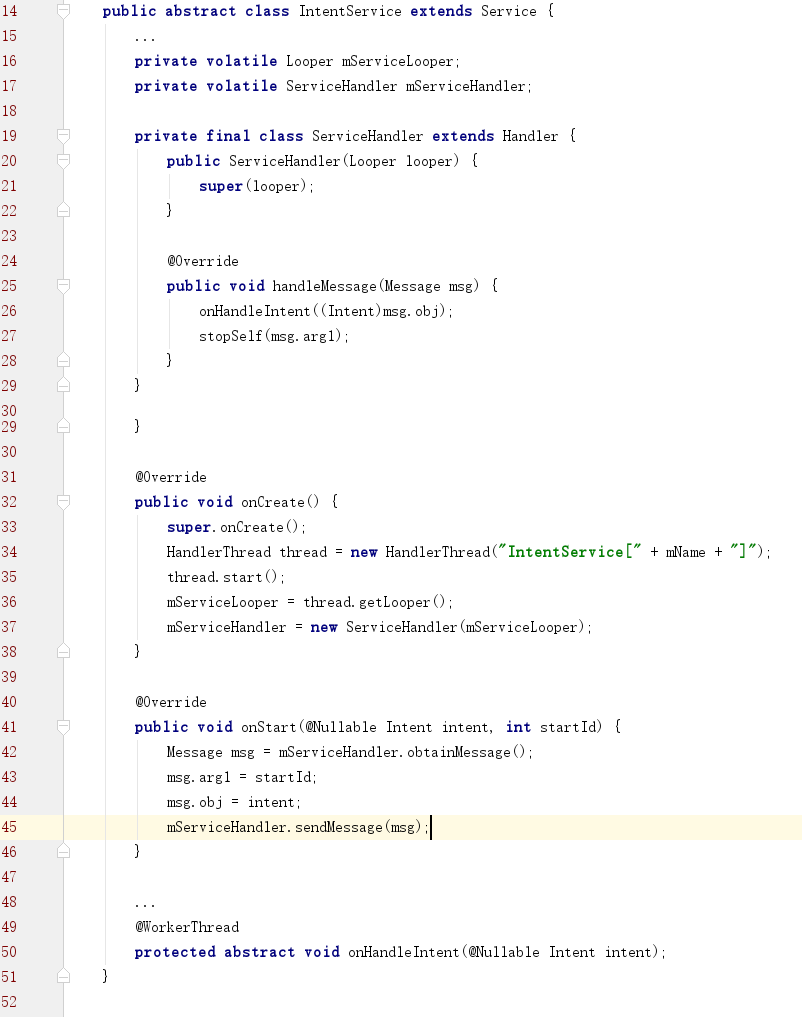
在run函数中，33行调用Looper类的静态成员函数prepare来准备一个消息循环对象。36行Looper类的myLooper成员函数将这个子线程中的消息循环对象保存在HandlerThread类中的成员变量mLooper中，这样，其它地方就可以方便地通过它的getLooper函数来获得这个消息循环对象了，有了这个消息循环对象后，就可以往这个子线程的消息队列中发送消息，通知这个子线程执行特定的任务了。41行在这个run函数通过Looper类的loop函数进入消息循环中。这样，一个具有消息循环的应用程序子线程就准备就绪了。

* **使用**



* **IntentService**

IntentService是一种特殊的service，它继承了service类并且是一个抽象类，因此必须创建它的子类才能使用它。由于IntentServiece是服务的原因，这导致它比单纯的线程的优先级要高，适合做一些高优先级的后台任务而不容易被系统杀死，当任务执行完后它会自动停止：

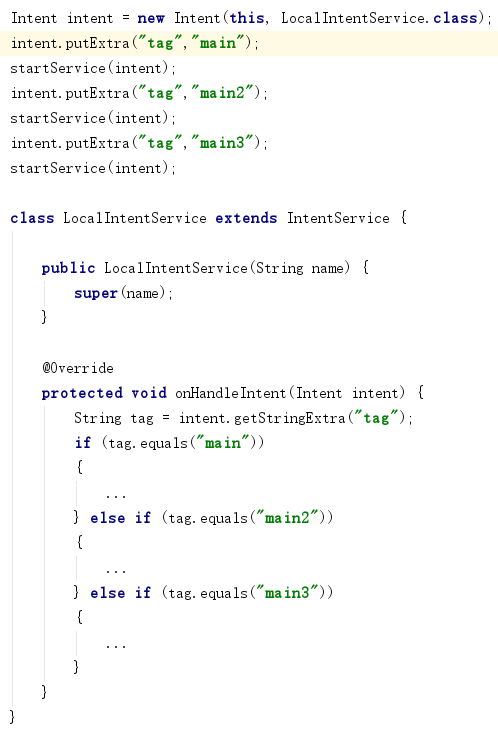


可以发现IntentService内部有一个常驻线程来执行耗时任务，当IntentService实例被创建的时候，在它的onCreate()方法中会创建并启动这个常驻线程，每次启动这个服务，都会调用onStart()方法来向常驻线程中发送一条消息，然后在25行handleMessage()方法中接收到消息并交给抽象方法onHandleIntent()来处理Intent，处理完后在27行调用stopSelf()来终止自己，在终止之前会等待所有的消息都处理完之后才会终止自己。

onHandleIntent()是一个抽象方法，它需要在子类中实现，它的作用是从Intent中区分具体的任务并执行这些任务。回调完成后会调用 stopSelf(msg.arg1)，注意这个msg.arg1是个int值，相当于一个请求的唯一标识。每发送一个请求，会生成一个唯一的标识，然后将请求放入队列，当全部执行完成(最后一个请求也就相当于getLastStartId == startId)，或者当前发送的标识是最近发出的那一个（getLastStartId == startId），则会销毁我们的Service。

如果传入的是-1则直接销毁。当任务完成销毁Service回调onDestory；

* **使用**



最后还需要在配置文件中配置：

<service android:name=**"com.work.project.LocalIntentService"**/>