IntentService是继承于Service并处理异步请求的一个类，在IntentService内有一个工作线程来处理耗时操作，启动IntentService的方式和启动传统Service一样，同时，当任务执行完后，IntentService会自动停止，而不需要我们去手动控制。另外，可以启动IntentService多次，而每一个耗时操作会以工作队列的方式在IntentService的onHandleIntent回调方法中执行，并且，每次只会执行一个工作线程，执行完第一个再执行第二个，以此类推。

而且，所有请求都在一个单线程中，不会阻塞应用程序的主线程（UI Thread），同一时间只处理一个请求。

### **IntentService有什么好处呢？**

1）我们省去了在Service中手动开线程的麻烦，

2）当操作完成时，我们不用手动停止Service。

接下来让我们来看看如何使用，写一个Demo来模拟两个耗时操作，Operation1与Operation2，先执行1，2必须等1执行完才能执行2：

新建工程，新建一个继承IntentService的类，我这里是IntentServiceDemo.java

public class IntentServiceDemo extends IntentService {  
  
 public IntentServiceDemo() {  
 //必须实现父类的构造方法  
 super("IntentServiceDemo");  
 }  
  
 @Override  
 public IBinder onBind(Intent intent) {  
 System.out.println("onBind");  
 return super.onBind(intent);  
 }  
  
  
 @Override  
 public void onCreate() {  
 System.out.println("onCreate");  
 super.onCreate();  
 }  
  
 @Override  
 public void onStart(Intent intent, int startId) {  
 System.out.println("onStart");  
 super.onStart(intent, startId);  
 }  
  
  
 @Override  
 public int onStartCommand(Intent intent, int flags, int startId) {  
 System.out.println("onStartCommand");  
 return super.onStartCommand(intent, flags, startId);  
 }  
  
  
 @Override  
 public void setIntentRedelivery(boolean enabled) {  
 super.setIntentRedelivery(enabled);  
 System.out.println("setIntentRedelivery");  
 }  
  
 @Override  
 protected void onHandleIntent(Intent intent) {  
 //Intent是从Activity发过来的，携带识别参数，根据参数不同执行不同的任务  
 System.out.println("currentThread()=" + Thread.currentThread().getName());  
 String action = intent.getExtras().getString("param");  
 if (action.equals("oper1")) {  
 System.out.println("Operation1");  
 }else if (action.equals("oper2")) {  
 System.out.println("Operation2");  
 }  
  
 try {  
 Thread.sleep(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Override  
 public void onDestroy() {  
 System.out.println("onDestroy");  
 super.onDestroy();  
 }  
  
}

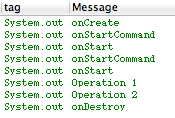
我把生命周期方法全打印出来了，待会我们来看看它执行的过程是怎样的。接下来是Activity，在Activity中来启动IntentService：

public class TestActivity extends Activity {  
 /\*\* Called when the activity is first created. \*/  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.main);  
  
 //可以启动多次，每启动一次，就会新建一个work thread，但IntentService的实例始终只有一个  
 //Operation 1  
 Intent startServiceIntent = new Intent("com.test.intentservice");  
 Bundle bundle = new Bundle();  
 bundle.putString("param", "oper1");  
 startServiceIntent.putExtras(bundle);  
 startService(startServiceIntent);  
  
 //Operation 2  
 Intent startServiceIntent2 = new Intent("com.test.intentservice");  
 Bundle bundle2 = new Bundle();  
 bundle2.putString("param", "oper2");  
 startServiceIntent2.putExtras(bundle2);  
 startService(startServiceIntent2);  
 }  
}

最后，别忘了配置Service，因为它继承于Service，所以，它还是一个Service，一定要配置，否则是不起作用的

<service android:name=".IntentServiceDemo">  
 <intent-filter >  
 <action android:name="com.test.intentservice"/>  
 </intent-filter>  
</service>

最后来看看执行结果:



img

从结果可以看到，onCreate方法只执行了一次，而onStartCommand和onStart方法执行了两次，开启了两个Work Thread，这就证实了之前所说的，启动多次，但IntentService的实例只有一个，这跟传统的Service是一样的。Operation1也是先于Operation2打印，并且我让两个操作间停顿了2s，最后是onDestroy销毁了IntentService。

### **IntentService 源码分析**

@Override  
public void onCreate() {  
 super.onCreate();  
 HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");  
 thread.start();  
 mServiceLooper = thread.getLooper();  
 mServiceHandler = new ServiceHandler(mServiceLooper);  
}

#### **源码可知：**

1）实际上是使用了一个 HandlerThread 来维护线程的，

2） HandleThread 中，内部已经维护一个 Looper，这里直接使用 HandlerThread 的 Looper 对象，便于在 IntentService 中去维护消息队列，

3）创建的 mServiceHandler 是属于 HandleThread 这个 WorkerThread 的。

private final class ServiceHandler extends Handler {  
 public ServiceHandler(Looper looper) {  
 super(looper);  
 }  
  
 @Override  
 public void handleMessage(Message msg) {  
 onHandleIntent((Intent)msg.obj);  
 stopSelf(msg.arg1);  
 }  
 }

#### **源码可知：**

1）直接把消息交给 onHandleIntent() 方法去执行具体的业务逻辑

2）执行完成之后，立即调用 stopSelf() 方法停止自己

接下来分析start源码

@Override  
 public void onStart(Intent intent, int startId) {  
 Message msg = mServiceHandler.obtainMessage();  
 msg.arg1 = startId;  
 msg.obj = intent;  
 mServiceHandler.sendMessage(msg);  
 }  
 @Override  
 public int onStartCommand(Intent intent, int flags, int startId) {  
 onStart(intent, startId);  
 return mRedelivery ? START\_REDELIVER\_INTENT : START\_NOT\_STICKY;  
 }

#### 源码可知

1）在 onStartCommand() 中直接调用了 onStart() 方法

2）而上面 stopSelf() 方法使用的 startId 来停止当前的此次任务服务。

3）而 Service 如果被启动多次，就会存在多个 startId ，当所有的 startId 都被停止之后，才会调用 onDestory() 自我销毁。

我们在看看HandlerThread启动之后的源码

@Override  
public void run() {  
 mTid = Process.myTid();  
 Looper.prepare();  
 synchronized (this) {  
 mLooper = Looper.myLooper();  
 notifyAll();  
 }  
 Process.setThreadPriority(mPriority);  
 onLooperPrepared();  
 Looper.loop();  
 mTid = -1;  
 }

#### **源码可知**

run方法里面添加了锁，这也解释了为什么多次 start 同一个 IntentService 它会顺序执行，全部执行完成之后，再自我销毁。