简介

IntentService是一种特殊的Service，它继承了Service并且是一个抽象类。IntentService用于后台执行耗时的任务，当任务执行后会自动停止。同时由于IntentService是服务的原因，它的优先级比单纯的线程要高很多。在实现上，IntentService封装了HandlerThread和Handler。这一点从它的onCreate方法可以看出来。

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

当IntentService第一次被启动时，onCreate方法会被调用。onCreate方法会创建一个HandlerThread，然后用它的Looper来构造一个Handler对象mServiceHandler，这样通过mServiceHandler发送消息最终会在HandlerThread的线程中执行。每次启动IntentService，都会调用onStartCommand方法，IntentService在onStartCommand中处理每个后台任务的Intent。onStartCommand方法又会调用onStart方法。

```
public void onStart(Intent intent, int startId) {
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    msg.obj = intent;
    mServiceHandler.sendMessage(msg);
}
```

可以看出，IntentService仅仅是通过mServiceHandler发送了一个消息，这个消息会在HandlerThread中被处理。mServiceHandler收到消息后，会将Intent对象传递给IntentService的onHandleIntent方法去处理。这里的Intent和外界startService(intent)中的intent完全一致。通过这个intent就可以解析外界传递的参数。当onHandleIntent方法执行结束后，IntentService会通过stopSelf(int startId)方法来尝试停止服务。这里不适用stopSelf()方法，是因为stopSelf()会立刻停止服务，这个时候可能还有其他消息未处理。stopSelf(int startId)则会等待所有的消息都处理完毕后才终止服务。一般来说，stopSelf(int startId)在尝试停止服务前，会判断最近启动服务的次数是否和startId相等，如果相等就立即停止服务，不相等就不停止服务。这个策略可以从AMS的stopServiceToken方法的实现中找到依据。ServiceHandler的实现如下：

```
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);
    }
}
```

另外，每执行一个后台任务就必须启动一次IntentService，而IntentService内部则通过消息的方式向HandlerThread请求执行任务，Handler中的Looper是顺序处理消息的，这就意味着IntentService也是顺序执行后台任务的。

使用示例

1. 派生一个IntentService的子类LocalIntentService

```
public class LocalIntentService extends IntentService {
    private static final String TAG = "LocalIntentService";

    public LocalIntentService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        String action = intent.getStringExtra("task_action");
        Log.d(TAG, "receive task :" +  action);
        SystemClock.sleep(3000);
        if ("com.ryg.action.TASK1".equals(action)) {
            Log.d(TAG, "handle task: " + action);
        }
    }

    @Override
    public void onDestroy() {
        Log.d(TAG, "service destroyed.");
        super.onDestroy();
    }
}
```

LocalIntentService在onHandleIntent方法中从Intent参数解析后台任务的标识，即task_action字段的内容。通过SystemClock.sleep(3000)来模拟耗时的后台任务。另外为了验证IntentService的停止时机，在onDestroy()中打印了日志。

2. 启动LocalIntentService开启后台任务

```
Intent service = new Intent(this, LocalIntentService.class);
service.putExtra("task_action", "com.ryg.action.TASK1");
startService(service);
service.putExtra("task_action", "com.ryg.action.TASK2");
startService(service);
service.putExtra("task_action", "com.ryg.action.TASK3");
startService(service);
```

运行后日志如下：

```
05-17 17:08:23.186 E/dalvikvm(25793): threadid=11: calling run(),name=
IntentService[LocalIntentService]
05-17 17:08:23.196 D/LocalIntentService(25793): receive task :com.ryg.
action.TASK1
05-17 17:08:26.199 D/LocalIntentService(25793): handle task: com.ryg.
action.TASK1
05-17 17:08:26.199 D/LocalIntentService(25793): receive task :com.ryg.
action.TASK2
05-17 17:08:29.192 D/LocalIntentService(25793): receive task :com.ryg.
action.TASK3
05-17 17:08:32.205 D/LocalIntentService(25793): service destroyed.
05-17 17:08:32.205 E/dalvikvm(25793): threadid=11: exiting,name=
IntentService[LocalIntentService]
```