我们知道Service里面不能执行太耗时的操作，耗时操作就需要创建自己的工作线程来执行，那么就可以使用IntentService

首先，定义一个TestIntentService继承自IntentService，并且重写onHandleIntent(Intent intent)方法，然后需要实现一个无参的构造函数:

//定义TestIntentService

public class TestIntentService extends IntentService {

private static final String TAG = "TestIntentService";

//应该提供一个无参构造，然后调用父类的有参构造

public TestIntentService() {

super("TestIntentService");

}

@Override

public void onCreate() {

super.onCreate();

//这个是在主线程的

Log.d(TAG, "onCreate: " + Thread.currentThread().getName());

}

@Override

protected void onHandleIntent(@Nullable Intent intent) {

//这个就在子线程了，也就是IntentService创建的工作线程

Log.d(TAG, "onHandleIntent: " + Thread.currentThread().getName());

if (intent == null) {

return;

}

//取出action

String action = intent.getAction();

//然后根据不同的action取出不同的参数来搞事情

if ("download".equals(action)) {

//播放

String url = intent.getStringExtra("url");

Log.d(TAG, "onHandleIntent, url=[" + url + "]");

} else if ("play".equals(action)) {

//下载

String id = intent.getStringExtra("id");

}

}

//当所有事情搞完了就会自动调到onDestroy()里面来

@Override

public void onDestroy() {

super.onDestroy();

//这个是在主线程的

Log.d(TAG, "onDestroy: " + Thread.currentThread().getName());

}

}

//使用

Intent intent = new Intent(this, TestIntentService.class);

intent.setAction("download");

intent.putExtra("url", "http://123.45.67.1/xxx/xxx");

startService(intent); //这里调用了就会跑到TestIntentService的onHandleIntent(Intent intent)中去

我们跑完上述代码就会发现打印如下日志:

D/TestIntentService: onCreate: main

D/TestIntentService: onHandleIntent: IntentService[TestIntentService]

D/TestIntentService: onHandleIntent, url=[http://123.45.67.1/xxx/xxx]

D/TestIntentService: onDestroy: main

说明它的流程就是:startService() (随便在哪个线程) -> onCreate (主线程) -> onHandleIntent() (工作线程) -> 跑自己的业务代码 (工作线程) -> onDestroy() (主线程)

public abstract class IntentService extends Service {

private volatile Looper mServiceLooper; //内部有个Looper

private volatile ServiceHandler mServiceHandler; //内部有个Handler

private String mName; // 名字，构造函数传递的

private boolean mRedelivery; //是否重发

//定义了自己的Handler

private final class ServiceHandler extends Handler {

public ServiceHandler(Looper looper) {

super(looper);

}

@Override

public void handleMessage(Message msg) {

//哦，在这里调用了onHandleIntent()，找到了

onHandleIntent((Intent)msg.obj); //这里取出了obj，强转为Intent，哪里发送?

//之后又就stopSelf()了，难怪刚刚我们打印完日志它就自己调到onDestroy()里面了

stopSelf(msg.arg1);

}

}

//这个参数是作为工作线程的名字用来调试用的

public IntentService(String name) {

super();

mName = name;

}

//如果设置为true，那么如果在onHandleIntent()期间进程死掉了，就会重启进程，并且再次发送一个Intent，如果有多个Intent，则只保证最近的一个能发出去

public void setIntentRedelivery(boolean enabled) {

mRedelivery = enabled;

}

@Override

public void onCreate() {

super.onCreate();

//在这里创建了一个HandlerThread，这个名字不就是我们刚刚传递的那个"TestIntentService"吗

HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");

//然后启动

thread.start();

//成员变量赋值，上一章我们分析过，如果取不到，会卡在这里

mServiceLooper = thread.getLooper();

mServiceHandler = new ServiceHandler(mServiceLooper);

}

@Override

public void onStart(@Nullable Intent intent, int startId) {

//哦，原来在这里构造Message并发出去

Message msg = mServiceHandler.obtainMessage();

msg.arg1 = startId;

msg.obj = intent; //obj就是Intent，刚刚那个强转是安全的

mServiceHandler.sendMessage(msg); //发射

}

@Override

public int onStartCommand(@Nullable Intent intent, int flags, int startId) {

onStart(intent, startId);

//根据是否重发来返回不同的flag

return mRedelivery ? START\_REDELIVER\_INTENT : START\_NOT\_STICKY;

}

@Override

public void onDestroy() {

//stopSelf()后调到这里，就退出Looper了，也就等价于退出了Thread了，玩完就扔，可以

mServiceLooper.quit();

}

@Override

@Nullable

public IBinder onBind(Intent intent) {

return null;

}

//使用模版模式，定义顶层抽象的onHandleIntent()提供给下层实现，顶层自己决定调用时机，可以

@WorkerThread

protected abstract void onHandleIntent(@Nullable Intent intent);

}

1 startService()后，就跑到了IntentService()的onCreate() 2 在onCreate()里面创建一个HandlerThread，然后取出Looper，并且用来创建自己的ServiceHandler 3 然后在onStart(Intent)里面构造消息，并且发送出去，这里的msg.obj = intent 3 在ServiceHandler里面的handleMessage()里面调用1 onHandleIntent(Intent(msg.obj)) 2 stopSelf()，注意这个handler是工作线程的，也就是说:已经切换到工作线程了 4 在onHandleIntent(intent)里面处理自己的业务逻辑，参数就从intent里面取 5 onHandleIntent()跑完了，接着跑stopSelf()，就跑到自己的onDestroy()里面去 6 在onDestroy()里面调用了mServiceLooper.quit()，就退出了工作线程