Notzuonotdied 调整IntentService文章中的编辑细节 4106f64 9 days ago

2 contributors

252 lines (184 sloc) 8.27 KB

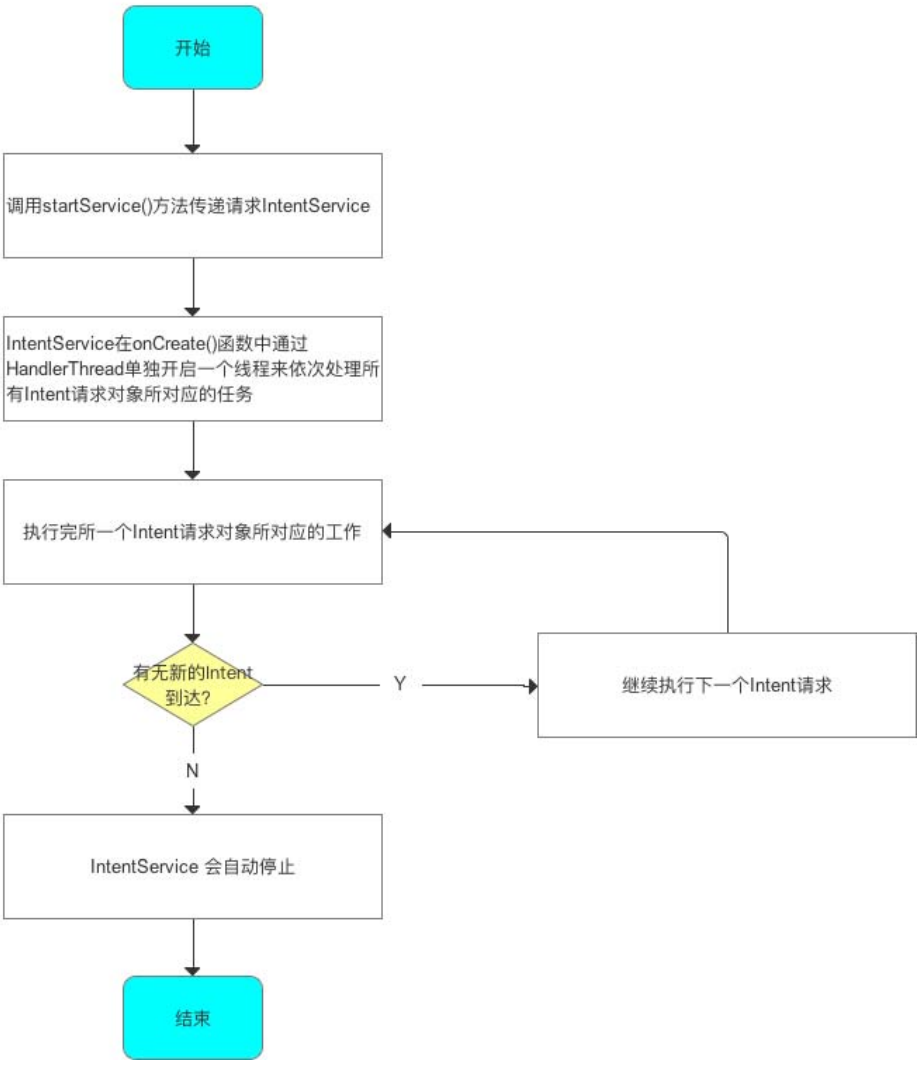
一、 定义

IntentService是Android里面的一个封装类，继承自四大组件之一的Service。

二、作用

处理异步请求，实现多线程。

三、 工作流程



注意：若启动IntentService多次，那么每个耗时操作则以队列的方式在IntentService的onHandleIntent回调方法中依次执行，执行完自动结束。

四、实现步骤

- 步骤1：定义IntentService的子类：传入线程名称、复写onHandleIntent()方法
- 步骤2：在Manifest.xml中注册服务
- 步骤3：在Activity中开启Service服务

五、具体实例

- 步骤1：定义IntentService的子类：传入线程名称、复写onHandleIntent()方法

```
package com.example.carson_ho.demoforintentservice;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

/**
 * Created by Carson_Ho on 16/9/28.
 */
public class myIntentService extends IntentService {

    /*构造函数*/
    public myIntentService() {
        //调用父类的构造函数
        //构造函数参数=工作线程的名字
        super("myIntentService");

    }

    /*复写onHandleIntent()方法*/
    //实现耗时任务的操作
    @Override
    protected void onHandleIntent(Intent intent) {
        //根据Intent的不同进行不同的事务处理
        String taskName = intent.getExtras().getString("taskName");
        switch (taskName) {
            case "task1":
                Log.i("myIntentService", "do task1");
                break;
            case "task2":
                Log.i("myIntentService", "do task2");
                break;
            default:
                break;
        }
    }


    @Override
    public void onCreate() {
        Log.i("myIntentService", "onCreate");
        super.onCreate();
    }

    /*复写onStartCommand()方法*/
    //默认实现将请求的Intent添加到工作队列里
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("myIntentService", "onStartCommand");
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        Log.i("myIntentService", "onDestroy");
        super.onDestroy();
    }
}
```

- 步骤2：在Manifest.xml中注册服务

```
<service android:name=".myIntentService">
        <intent-filter>
                <action android:name="cn.scu.finch"/>
```

```
        </intent-filter>
</service>
```

- 步骤3：在Activity中开启Service服务

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

        //同一服务只会开启一个工作线程
        //在onHandleIntent函数里依次处理intent请求。

        Intent i = new Intent("cn.scu.finch");
        Bundle bundle = new Bundle();
        bundle.putString("taskName", "task1");
        i.putExtras(bundle);
        startService(i);

        Intent i2 = new Intent("cn.scu.finch");
        Bundle bundle2 = new Bundle();
        bundle2.putString("taskName", "task2");
        i2.putExtras(bundle2);
        startService(i2);

        startService(i);  //多次启动
    }
}
```

- 结果

Tag	Text
myIntentService	onCreate
myIntentService	onStartCommand
myIntentService	onStartCommand
myIntentService	do task1
myIntentService	onStartCommand
myIntentService	do task2
myIntentService	do task1
myIntentService	onDestroy

六、源码分析

接下来，我们会通过源码分析解决以下问题：

- IntentService如何单独开启一个新的工作线程；
- IntentService如何通过onStartCommand()传递给服务intent被**依次**插入到工作队列中

问题1：IntentService如何单独开启一个新的工作线程

```
// IntentService源码中的 onCreate() 方法
@Override
public void onCreate() {
    super.onCreate();
    // HandlerThread继承自Thread，内部封装了 Looper
    //通过实例化HandlerThread新建线程并启动
    //所以使用IntentService时不需要额外新建线程
    HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");
    thread.start();

    //获得工作线程的 Looper，并维护自己的工作队列
    mServiceLooper = thread.getLooper();
    //将上述获得Looper与新建的mServiceHandler进行绑定
    //新建的Handler是属于工作线程的。
    mServiceHandler = new ServiceHandler(mServiceLooper);
}

private final class ServiceHandler extends Handler {

    public ServiceHandler(Looper looper) {
        super(looper);
```

```
    }

    // IntentService的handleMessage方法把接收的消息交给onHandleIntent()处理
    // onHandleIntent()是一个抽象方法，使用时需要重写的方法
    @Override
    public void handleMessage(Message msg) {
        // onHandleIntent 方法在工作线程中执行，执行完调用 stopSelf() 结束服务。
        onHandleIntent((Intent) msg.obj);
        //onHandleIntent 处理完成后 IntentService会调用 stopSelf() 自动停止。
        stopSelf(msg.arg1);
    }
}

// onHandleIntent()是一个抽象方法，使用时需要重写的方法
@WorkerThread
protected abstract void onHandleIntent(Intent intent);
```

问题2：IntentService如何通过onStartCommand()传递给服务intent被依次插入到工作队列中

```
public int onStartCommand(Intent intent, int flags, int startId) {
    onStart(intent, startId);
    return mRedelivery ? START_REDELIVER_INTENT : START_NOT_STICKY;
}

public void onStart(Intent intent, int startId) {
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    //把 intent 参数包装到 message 的 obj 中，然后发送消息，即添加到消息队列里
    //这里的Intent 就是启动服务时startService(Intent) 里的 Intent。
    msg.obj = intent;
    mServiceHandler.sendMessage(msg);
}

//清除消息队列中的消息
@Override
public void onDestroy() {
    mServiceLooper.quit();
}
```

- 总结

  从上面源码可以看出，IntentService本质是采用Handler & HandlerThread方式：

  i. 通过HandlerThread单独开启一个名为**IntentService**的线程
  ii. 创建一个名叫ServiceHandler的内部Handler
  iii. 把内部Handler与HandlerThread所对应的子线程进行绑定
  iv. 通过onStartCommand()传递给服务intent，**依次**插入到工作队列中，并逐个发送给onHandleIntent()
  v. 通过onHandleIntent()来依次处理所有Intent请求对象所对应的任务

因此我们通过复写方法onHandleIntent()，再在里面根据Intent的不同进行不同的线程操作就可以了

注意事项：工作任务队列是顺序执行的。

> 如果一个任务正在IntentService中执行，此时你再发送一个新的任务请求，这个新的任务会一直等待直到前面一个任务执行完毕才开始执行。

原因：

1. 由于onCreate() 方法只会调用一次，所以只会创建一个工作线程；
2. 当多次调用 startService(Intent) 时（onStartCommand也会调用多次）其实并不会创建新的工作线程，只是把消息加入消息队列中等待执行，**所以，多次启动 IntentService 会按顺序执行事件**；
3. 如果服务停止，会清除消息队列中的消息，后续的事件得不到执行。

七、使用场景

- 线程任务需要按顺序、在后台执行的使用场景

  > 最常见的场景：离线下载

- 由于所有的任务都在同一个Thread looper里面来做，所以不符合多个数据同时请求的场景。

八、对比

8.1 IntentService与Service的区别

- 从属性 & 作用上来说 Service：依赖于应用程序的主线程（不是独立的进程 or 线程）

  > 不建议在Service中编写耗时的逻辑和操作，否则会引起ANR；

  IntentService：创建一个工作线程来处理多线程任务

- Service需要主动调用stopSelft()来结束服务，而IntentService不需要（在所有intent被处理完后，系统会自动关闭服务）

8.2 IntentService与其他线程的区别

- IntentService内部采用了HandlerThread实现，作用类似于后台线程；
- 与后台线程相比，IntentService是一种后台服务，优势是：优先级高（不容易被系统杀死），从而保证任务的执行。

  > 对于后台线程，若进程中没有活动的四大组件，则该线程的优先级非常低，容易被系统杀死，无法保证任务的执行